

Extracting Common Sense Knowledge from Text for Robot Planning

Peter Kaiser¹ Mike Lewis² Ronald P. A. Petrick² Tamim Asfour¹ Mark Steedman²

Abstract—Autonomous robots often require domain knowledge to act intelligently in their environment. This is particularly true for robots that use automated planning techniques, which require symbolic representations of the operating environment and the robot’s capabilities. However, the task of specifying domain knowledge by hand is tedious and prone to error. As a result, we aim to automate the process of acquiring general common sense knowledge of objects, relations, and actions, by extracting such information from large amounts of natural language text, written by humans for human readers.

We present two methods for knowledge acquisition, requiring only limited human input, which focus on the inference of spatial relations from text. Although our approach is applicable to a range of domains and information, we only consider one type of knowledge here, namely object locations in a kitchen environment. As a proof of concept, we test our approach using an automated planner and show how the addition of common sense knowledge can improve the quality of the generated plans.

I. INTRODUCTION AND RELATED WORK

Autonomous robots that use automated planning to make decisions about how to act in the world require symbolic representations of the robot’s environment and the actions the robot is able to perform. Such models can be aided by the presence of *common sense knowledge*, which may help guide the planner to build higher quality plans, compared with the absence of such information. In particular, knowledge about default locations of objects (*the juice is in the refrigerator*) or the most suitable tool for an action (*knives are used for cutting*) could help the planner make decisions on which actions are more appropriate in a given context.

For example, if a robot needs a certain object for a task, it can typically employ one of two strategies in the absence of prior domain knowledge: the robot can ask a human for the location of the object, or the robot can search the domain in an attempt to locate the object itself. Both techniques are potentially time consuming and prevent the immediate deployment of autonomous robots in unknown environments. By contrast, the techniques proposed in this paper allow the robot to consider likely locations for an object, informed by common sense knowledge. This potentially improves plan quality, by avoiding exhaustive search, and does not require the aid of a human either to inform the robot directly or to encode the necessary domain knowledge a priori.

While it is not possible to automatically generate all the domain knowledge that could possibly be required, we propose two methods for learning useful elements of

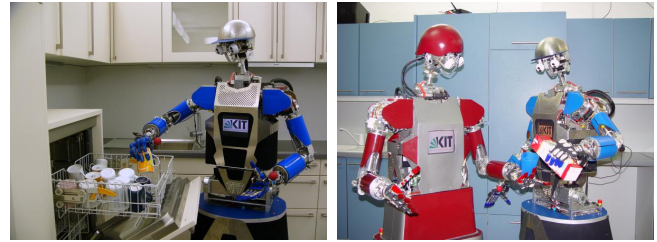


Fig. 1: The humanoid robots ARMAR-IIIa (left) and ARMAR-IIIb working in a kitchen environment ([5], [6]).

domain knowledge based on information gathered from natural language texts. These methods will provide the set of object and action types for the domain, as well as certain relations between entities of these types, of the kind that are commonly used in planning. As an evaluation, we build a domain for a robot working in a kitchen environment (see Fig. 1) and infer spatial relations between objects in this domain. We then show how the induced knowledge can be used by an automated planning system. (The generated symbols will not be grounded in the robot’s internal model; however, approaches to establish these links given names of objects or actions are available (e.g., [1], [2], [3] and [4]).)

The extraction of spatial relations from natural language has been studied in the context of understanding commands and directions given to robots in natural language (e.g., [7], [8], [9]). In contrast to approaches based on annotated corpora of command executions or route instructions, or the use of knowledge bases like *Open Mind Common Sense* [10] explicitly created for artificial intelligence applications, we extract relevant relations from large amounts of text written by humans for humans. The text mining techniques used in [11], [12], [13] to extract action-tool relations to disambiguate visual interpretations of kitchen actions are related. In [14], spatial relations are inferred based on search engine queries and common sense databases.

In the following, we describe a process for learning domain ontologies (Section II) and for extracting relations (Section III). The last two sections evaluate both methods (Section IV) and describe how the resulting knowledge can be used in an automated planning system (Section V).

II. AUTOMATIC DOMAIN ONTOLOGY LEARNING

In this section, we propose a method for automatically learning a *domain ontology* \mathcal{D} —a set of symbols that refer to a robot’s environment or capabilities—with very little human input. The method can be configured to learn a domain

¹Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology, Karlsruhe, Germany {peter.kaiser, asfour}@kit.edu

²School of Informatics, University of Edinburgh, Edinburgh, Scotland, United Kingdom {mike.lewis, rpetrick, steedman}@inf.ed.ac.uk

of objects or actions. With robotic planning in mind, it is crucial that in either of these cases, the contained symbols are not too abstract. In terms of a kitchen environment, interesting objects might be *saucepan*, *refrigerator* or *apple*, while abstract terms like *minute* or *temperature* that do not directly refer to objects are avoided. Similarly, we focus on actions that are directly applied to objects like *knead*, *open* or *screw*, and ignore more abstract actions like *have* or *think*.

Automatic domain ontology learning is based on a *domain-defining corpus* \mathcal{C}_D , which contains texts concerning the environment that the domain should model. For example, a compilation of recipes is a good domain-defining corpus for a kitchen environment. Note that these texts have been written by humans for human readers, and no efforts are taken to make them more suitable for \mathcal{C}_D . However, \mathcal{C}_D needs to be part-of-speech (POS) tagged and possibly parsed if compound nouns are to be appropriately recognized.¹

The domain-defining corpus \mathcal{C}_D is used to retrieve an initial vocabulary \mathcal{V} which is then filtered for abstract symbols. Depending on the type of symbol that this vocabulary is meant to model, only nouns or verbs are included in \mathcal{V} . In the first step, \mathcal{C}_D is analyzed for word frequency and the k most frequent words are extracted (see Alg. 1). Only words with a part-of-speech-tag (*POS-tag*) equal to $p \in \{\text{noun}, \text{verb}\}$ are considered. The resulting vocabulary \mathcal{V} is then filtered according to the score $\Theta(w, p)$ which expresses the *concreteness* of a word w .

Algorithm 1: $\text{learnDomainOntology}(\mathcal{C}_D, p, k, \Theta_{\min})$

- 1 $\mathcal{V} \leftarrow \text{mostFrequentWords}(\mathcal{C}_D, p, k)$
 - 2 $\mathcal{D} \leftarrow \{w \in \mathcal{V} : \Theta(w, p) \geq \Theta_{\min}\}$
 - 3 **return** \mathcal{D}
-

Fig. 2 gives an overview of the domain ontology learning process. Additionally, it shows details about the relation extraction procedure that will be discussed below, and the interoperability between the two methods. In the following section, we discuss the concreteness score Θ in detail.

A. The Concreteness Θ

Having a measure of concreteness is necessary for filtering symbols that are too abstract to play a role in our target domain. In particular, the score $\Theta(w, p)$ resembles the concreteness of a word w with POS-tag p using the lexical database *WordNet* ([16], [17]). For nouns, WordNet features an ontology that differs between physical and abstract entities. However, a word can have different meanings, some of which could be abstract and others not. WordNet solves this issue by working on word-senses rather than on words. For a word w with a sense² s from the set $S(w)$ of possible senses of w , we can compute a Boolean indicator $c_{w,s}$ that

¹We use the *Stanford Parser* [15] to do this.

²WordNet numbers the different word-senses, so $S(w) \subset \mathbb{N}$.

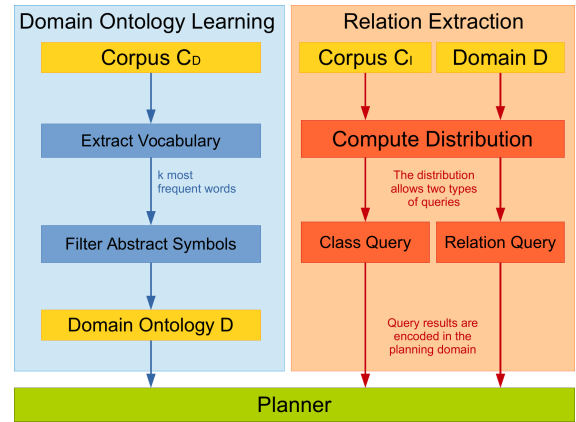


Fig. 2: The process of domain ontology learning (left) and relation extraction (right). The ontology resulting from the first method can be used as input to relation extraction.

tells us if s is a physical meaning of w :

$$c_{w,s} = \begin{cases} 1, & \text{if } s \text{ is a physical meaning of } w \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

WordNet also features a frequency measure $f_{w,s}$ that indicates how often a word w was encountered in the sense of s , based on a reference corpus. As we are not doing semantic parsing on \mathcal{C}_D , we do not know which of the possible senses of w is true. However, we can compute a weighted average of the concreteness of the different meanings of w , weighing each word-sense with its likelihood:

$$\Theta(w) = \frac{\sum_{s \in S(w)} f_{w,s} \cdot c_{w,s}}{\sum_{s \in S(w)} f_{w,s}}. \quad (2)$$

As a byproduct, Θ can only have nonzero values for words that are contained in WordNet, which filters out misspelled words or parsing errors.

As there is no suitable differentiation in WordNet’s ontology for verbs, we can not apply the exact same approach here. However, WordNet features a rough clustering of verbs that we use to define the filter. We set $c_{w,s}$ to 1, if the verb w with sense s is in one of the following categories: *verb.change*, *verb.contact*, *verb.creation* or *verb.motion*.

III. RELATION EXTRACTION

The second technique we propose for information acquisition deals with relations between symbols, defined using syntactic patterns. Such patterns capture the syntactic contexts that describe the relevant relations as well as the relations’ arguments. For example, the pattern

$$((\#object, noun), (in, prep), (\#location, noun)) \quad (3)$$

describes a prepositional relation between two nouns using the preposition *in*. The pattern also defines two classes,³ *#object* and *#location*, which stand for the two arguments of the relation. Given the above syntactic pattern, two types of questions are relevant in this work:

³In examples we use a hash to indicate a classname.

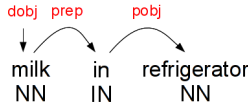


Fig. 3: A dependency path for the fragment *milk in refrigerator* contains the words, their respective POS-tag and the syntactic relation between them.⁴

- *Class inference*: Is a symbol w more likely an object or a location?
- *Relation inference*: What is the most likely location for a symbol w ?

The acquisition of relational information is interesting for endowing a robot with initial knowledge of its environment. Our main application for this method is the extraction of spatial relations in a kitchen setting, such as the location of common objects. However, the proposed method is not constrained to objects and locations, and we will show different use cases in the evaluation in Section IV-C.

The relation extraction process works in two phases:

- In the *crawling phase*, the text sources are searched for predefined syntactic patterns. Words falling into classes defined in the patterns are counted. The counts are compiled to a set of distributions.
- In the *query phase*, information can be queried from the distributions computed in the crawling phase. Different kinds of queries are possible.

The foundation for relation extraction is the *domain-independent corpus* \mathcal{C}_I . In contrast to the domain-defining corpus \mathcal{C}_D , \mathcal{C}_I contains unrestricted text. Because it is rare for common sense information to be explicitly expressed, the size of \mathcal{C}_I is crucial. We assume that the domain-independent corpus is dependency parsed, i.e., consists of syntactic dependency paths of the kind shown in Fig. 3. A further discussion of \mathcal{C}_I is given in Section IV-C.

In the following sections, we give a formal definition of a syntactic pattern and explain the two phases in further detail. Fig. 2 gives an overview of relation extraction.

A. Syntactic Patterns

The goal of the crawling phase is to search large amounts of texts for syntactic patterns predefined by the user. These patterns are designed to specify a relation between classes of words. For example, pattern (3) describes a spatial relation between the two classes *#object* and *#location*. The fragment *milk in refrigerator* would match the pattern and would result in the assignments *#object=milk* and *#location=refrigerator*.

Formally, a syntactic pattern is defined as a sequence of tuples containing a symbol s_i and a POS-tag p_i :

$$\Pi = ((s_1, p_1), \dots, (s_k, p_k)). \quad (4)$$

When matching the pattern to a sequence of words, the tuples will match exactly one word of the sequence. The condition for a match depends on the symbol s_i :

⁴*NN* - noun, *IN* - preposition
dobj - direct object, *prep* - preposition, *pobj* - prepositional object

- If s_i is a word, the i -th tuple matches to this exact word with POS-tag p_i .
- If s_i is a classname, the i -th tuple matches to all words from \mathcal{D} having the POS-tag p_i .

We will use the predicates $isclass(s_i)$ and $isword(s_i)$ to differ between the two possible meanings of the symbol s_i .

The search for matches happens on word-sequences. Such sequences can represent sentences or, as is the case for our corpus \mathcal{C}_I , dependency paths. A word-sequence contains words w_i together with their respective POS-tag t_i :

$$\Sigma = ((w_1, t_1), \dots, (w_n, t_n)), \quad (5)$$

Alg. 2 decides if an element (s, p) from a syntactic pattern matches an element (w, t) from a word-sequence. If the symbol s is a class, it only checks if the word w is part of the domain ontology \mathcal{D}_p that contains the valid words with POS-tag p . If s is a word, it must equal w . In both cases, the POS-tags p and t have to match.

Algorithm 2: $match((s, p), (w, t), \mathcal{D})$

```

1 if  $isclass(s)$  then
2   | return  $p = t \wedge w \in \mathcal{D}_p$ 
3 end
4 else if  $isword(s)$  then
5   | return  $p = t \wedge s = w$ 
6 end

```

Alg. 3 describes the matching process for a complete syntactic pattern (using Alg. 2). If a match is found, the class configuration is returned as a set of class assignments, i.e., class-word pairs. For example, using pattern (3), the fragment *milk in refrigerator* results in the class configuration:

$$\mathcal{K} = \{(\#object, milk), (\#location, refrigerator)\}. \quad (6)$$

Algorithm 3: $configuration(\Sigma, \Pi, \mathcal{D})$

```

1 for  $i = 1, \dots, |\Sigma| - |\Pi| + 1$  do
2   |  $\mathcal{I} \leftarrow \{0, \dots, |\Pi| - 1\}$ 
3   | if  $match(\Pi_{j+1}, \Sigma_{i+j}, \mathcal{D}) \forall j \in \mathcal{I}$  then
4     | | return  $\{(s_{j+1}, w_{i+j}) : j \in \mathcal{I}, isclass(s_{j+1})\}$ 
5     | end
6 end
7 return  $\emptyset$ 

```

B. The Crawling Phase

In the crawling phase, \mathcal{C}_I is searched for pattern matches using Alg. 2 and Alg. 3. Two different distributions are then computed based on the resulting class configurations:

- The *Relation Distribution* D_R counts the occurrences of class configurations (e.g., (6)). D_R is suitable for answering the question: How likely is a class configuration for the relation induced by pattern Π ?

- The *Class Distribution* D_C counts the occurrences of individual class assignments. It is suitable for answering the question: How likely is a class for a given word?

Alg. 4 shows how D_R and D_C are computed given a set of dependency paths \mathcal{S} and a syntactic pattern Π .

Algorithm 4: *computeDistribution*($\mathcal{S}, \Pi, \mathcal{D}$)

```

1  $D_R \leftarrow$  Empty Relation Distribution
2  $D_C \leftarrow$  Empty Class Distribution
3 foreach  $\Sigma = ((w_1, t_1), \dots, (w_n, t_n)) \in \mathcal{S}$  do
4    $\mathcal{K} \leftarrow$  configuration( $\Sigma, \Pi, \mathcal{D}$ )
5   if  $\mathcal{K} \neq \emptyset$  then
6      $D_R[\mathcal{K}] \leftarrow D_R[\mathcal{K}] + 1$ 
7     foreach  $(c, w) \in \mathcal{K}$  do
8        $D_C[(c, w)] \leftarrow D_C[(c, w)] + 1$ 
9     end
10  end
11 end
12 return ( $D_R, D_C$ )

```

C. The Query Phase

The query phase uses the distributions D_R and D_C to compute pseudo-probabilities for class assignments.

A *class query* $\gamma(c, w)$ approximates the probability of a word w falling into a class c . If $\Gamma = \{c_1, \dots, c_l\}$ is the set of defined classes, the class query can be formulated as:

$$\gamma(c, w) = \frac{D_C[(c, w)]}{\sum_{x \in \Gamma} D_C[(x, w)]}. \quad (7)$$

A *relation query* $\rho(Q, c_*)$ approximates the probability of a relation with class-assignments $Q = \{(c_1, w_1), \dots, (c_l, w_l)\}$, normalizing over the possible values of the class c_* . With $Q_* = \{(c, w) \in Q : c \neq c_*\}$, the relation query can be formulated as:

$$\rho(Q, c_*) = \frac{D_R[Q]}{\sum_{v \in \mathcal{D}} D_R[Q_* \cup \{(c_*, v)\}]}. \quad (8)$$

In the evaluation, we will consider both types of queries.

IV. EVALUATION

To evaluate the proposed methods of domain learning and relation extraction, we first show that it is possible to use a specialized corpus to generate a domain ontology of entity types that matches people’s expectations for the kitchen environment. We then use another, more general text corpus, to infer spatial relations and action-tool relations for those entities. These components are independent: we show in Section V that hand specification of the domain entities by a human expert can aid the automated extraction processes.

A. Prerequisites

To learn a domain ontology using the proposed method, the two text corpora \mathcal{C}_D and \mathcal{C}_I must first be defined.

1) *The domain-defining Corpus* \mathcal{C}_D : This corpus is used to generate an initial vocabulary by analysing word frequencies. \mathcal{C}_D should therefore be reasonably large but, more importantly, should contain descriptions of common objects and actions from the desired domain. For a kitchen environment, we chose to build \mathcal{C}_D from a set of about 11,000 recipes,⁵ with a total size of 19.5 MB.

2) *The domain-independent Corpus* \mathcal{C}_I : The domain-independent corpus is used to sort entities into different classes according to the results of syntactic pattern matches. \mathcal{C}_I does not need to be a different corpus than \mathcal{C}_D , but it is difficult to extract reliable information on rare relations from small corpora. This is especially true for common sense knowledge that isn’t often explicitly expressed. Hence, \mathcal{C}_I should be extensive. As it is often difficult to gather large amounts of text about a specific topic, it is useful to separate \mathcal{C}_I from \mathcal{C}_D , and use a large standard corpus for \mathcal{C}_I .

We use the *Google Books Ngrams Corpus* [18], in the following referred to as the *Google Corpus*, which contains a representation of 3.5 million English books containing about 345 billion words in total. The corpus is already parsed, tagged and frequency counted. The Google Corpus does not work on sentences, but on *syntactic ngrams* (Fig. 3), which are n content-word long subpaths of the dependency paths. We use the corpus in its *arcs* form which contains syntactic ngrams with two content-words ($n = 2$) plus possible non-content-words like prepositions or conjunctions. However, the proposed methods can also be used in combination with corpora containing longer syntactic ngrams.

B. Domain Ontology Learning

Using the corpora mentioned above, we can run the method for automatic domain ontology learning. Generating a domain ontology for nouns using parameter values of $k = 300$, $\Theta_{\min} = 0.35$ results in an ontology of 198 words of which the 80 most frequent ones are listed in Table I. The 20 most frequent nouns that were part of the initial vocabulary, but did not pass the concreteness filter are listed in Table II.

Analogously, the 80 most frequent actions from a domain ontology learnt from verbs using the parameters $k = 300$, $\Theta_{\min} = 0.2$ are depicted in Table III. The full domain ontology contains 206 verbs. The 20 most frequent verbs that did not pass the concreteness filter are listed in Table IV.

Results show that for objects, as well as actions, the generated domain ontologies are reasonable, but contain obvious mistakes. For example, the concrete noun *cream* was rejected while abstract nouns like *top* and *bottom* were included. The reason for this is the diversity of possible word senses present in WordNet that can mislead the filter Θ .

To evaluate the strength of the domain learning method, we asked four people⁶ to manually extract kitchen-related objects and actions from the sets of the 300 most frequent nouns and verbs from \mathcal{C}_D . Fig. 4 shows the F_1 -scores of the automatically learnt domain ontologies for objects and

⁵From <http://www.ewh.com>.

⁶Native speakers of English, not involved in the research.

TABLE I: Automatic domain ontology learning (objects)

$k = 300, \Theta_{\min} = 0.35$

1	wine	21	milk	41	container	61	salad
2	water	22	bottle	42	home	62	tea
3	meat	23	fruit	43	bag	63	grill
4	bowl	24	pot	44	garlic	64	center
5	sugar	25	dough	45	skillet	65	soup
6	mixture	26	glass	46	hand	66	alcohol
7	pan	27	side	47	lid	67	coffee
8	oil	28	pepper	48	onion	68	beer
9	top	29	meal	49	skin	69	sheet
10	oven	30	flour	50	saucepan	70	world
11	salt	31	fish	51	egg	71	diet
12	dish	32	refrigerator	52	beef	72	freezer
13	cheese	33	drink	53	layer	73	blender
14	cup	34	chocolate	74	piece	89	batter
15	butter	35	turkey	55	liquid	75	pasta
16	chicken	36	bottom	56	spoon	76	pork
17	juice	37	cake	57	surface	77	addition
18	bread	38	place	58	restaurant	78	dinner
19	rice	39	ice	59	fat	79	vodka
20	sauce	40	knife	60	plate	80	powder

TABLE II: Not part of the object domain ontology

$k = 300, \Theta_{\min} = 0.35$

1	time	6	taste	11	temperature	16	type
2	flavor	7	way	12	day	17	tbsp
3	heat	8	variety	13	process	18	color
4	recipe	9	cream	14	boil	19	hour
5	food	10	amount	15	cooking	20	half

actions, using different values for Θ_{\min} , compared to the domains created by the human participants. The results show that enabling the concreteness filter ($\Theta_{\min} > 0$) significantly increases the quality of the resulting domain for nouns as well as for verbs. The results also show that values of roughly $\Theta_{\min} > 0.5$ result in a too restrictive filter. While in the case of nouns, the restrictive filter still produces a better domain than if no filter is applied, this is not true for verbs: the quality of a verb-domain drops dramatically the more restrictive the filter gets. The reason for the difference between the two plots is that verbs often have a variety of possible meanings. By contrast, nouns usually have a predominant interpretation, at least in terms of the differentiation between physical and abstract meanings. This is also reflected in the fact that the participants found it significantly harder to create a domain of actions than to create a domain of nouns.

C. Inference

We now evaluate the relation and class inference mechanisms described in Section III. To illustrate the capabilities of these methods, we generated the results using the manually created domain ontology as a gold standard. We additionally show how false positives can affect the process by using the automatically learnt domain ontology. The parameter Θ_{\min} can be determined in practice by generating and evaluating an ontology for a subset of the initial vocabulary using plots similar to Fig. 4. Different syntactic patterns can be used to conduct different kinds of inference. The following sections show examples of possible queries.

1) *Location Inference*: A good use of knowledge acquisition is the exploration of spatial relations between objects

TABLE III: Automatic domain ontology learning (actions)

$k = 300, \Theta_{\min} = 0.2$

1	add	21	cool	41	come	61	stick
2	make	22	fill	42	press	62	beat
3	place	23	leave	43	freeze	63	clean
4	remove	24	go	44	garnish	64	begin
5	cook	25	bring	45	pick	65	burn
6	pour	26	hold	46	open	66	spread
7	stir	27	reduce	47	slice	67	replace
8	do	28	follow	48	become	68	whisk
9	put	29	heat	49	refrigerate	69	boil
10	take	30	pan	50	soak	70	produce
11	get	31	sprinkle	51	dip	71	preheat
12	turn	32	dry	52	form	72	squeeze
13	set	33	start	53	shake	73	chill
14	cut	34	melt	74	cause	89	top
15	cover	35	sit	55	pull	75	peel
16	mix	36	chop	56	break	76	fit
17	combine	37	drain	57	wash	77	move
18	create	38	rinse	58	simmer	78	coat
19	prepare	39	blend	59	lay	79	increase
20	bake	40	roll	60	transfer	80	seal

TABLE IV: Not part of the action domain ontology

$k = 300, \Theta_{\min} = 0.2$

1	be	6	keep	11	eat	16	check
2	use	7	let	12	choose	17	enjoy
3	have	8	allow	13	need	18	give
4	serve	9	try	14	help	19	see
5	show	10	find	15	buy	20	want

and locations using prepositional contexts. For instance, pattern (3) matches fragments where two nouns, *#object* and *#location*, are linked by the preposition *in*. This pattern can be used in combination with the above object ontology (Table I) to infer spatial relations in a kitchen environment. Table V shows the most likely locations for the ten most frequent objects from the automatically learnt domain ontology.⁷ Note that for generating the results we used pattern (3) combined with three similar patterns using the prepositions *on*, *at* and *from*. Table V presents two sets of locations for each object: the upper, highlighted rows refer to the manually created domain ontology and the lower, non-highlighted rows refer to the automatically learnt domain ontology in Table I.

Results from the automatically learnt domain ontologies are more noisy, and distractive terms like *side* or *bottom* haven't been filtered out. (We can also tune domain generation to work in a more restrictive way, e.g., by using the $F_{0.5}$ measure instead of F_1 to emphasize precision over recall.)

The results demonstrate that the system is able to infer typical locations for objects. However, two problems constrain its performance. First, the automatically learnt domain ontology does not contain typical locations like *cupboard* or *drawer*, because these words do not frequently appear in the initial vocabulary. Second, the system is not able to differ between container objects like *pot* or *pan*, and actual locations like *refrigerator* or *oven* (i.e., objects that have a fixed position in the kitchen). Improving the domain entity specification by using more diverse but relevant domain specific corpora is the subject of ongoing research. In Section V

⁷We consider *top* and *oven* not to be objects.

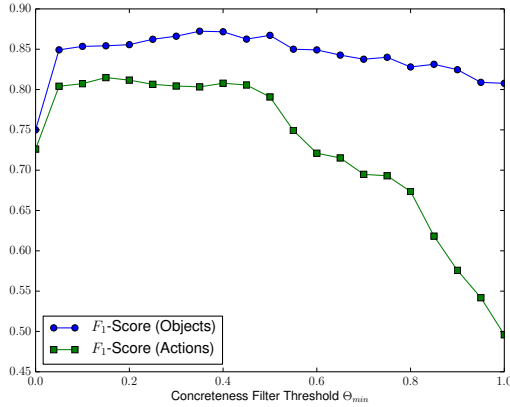


Fig. 4: Automatically learnt domain ontologies are evaluated using different values of Θ_{\min} , by comparing them to domain ontologies created manually by human participants.

TABLE V: Results for location inference

Highlighted rows: Manually created domain ontology. Non-highlighted rows: Automatically learnt domain ontology.

refr. - refrigerator, scp. - saucepan, swc. - sandwich, kit. - kitchen

object	first	second	third	fourth
wine	glass / 0.20	bottle / 0.20	table / 0.15	cup / 0.14
	glass / 0.13	bottle / 0.13	table / 0.10	cup / 0.09
water	surface / 0.09	bottle / 0.07	water / 0.07	glass / 0.07
	bottom / 0.06	side / 0.06	surface / 0.05	bottle / 0.04
meat	table / 0.08	pan / 0.06	swc. / 0.05	pot / 0.05
	diet / 0.11	table / 0.05	pan / 0.04	swc. / 0.04
bowl	table / 0.51	refr. / 0.06	stem / 0.05	kit. / 0.05
	table / 0.27	hand / 0.20	top / 0.06	side / 0.05
sugar	water / 0.15	bowl / 0.15	scp. / 0.10	milk / 0.08
	water / 0.13	bowl / 0.13	scp. / 0.08	milk / 0.06
mixture	pan / 0.10	bowl / 0.08	water / 0.08	dish / 0.08
	top / 0.10	pan / 0.07	bowl / 0.05	water / 0.05
pan	oven / 0.40	stove / 0.19	rack / 0.18	pan / 0.02
	oven / 0.29	stove / 0.14	rack / 0.13	hand / 0.07
oil	skillet / 0.22	pan / 0.19	scp. / 0.08	board / 0.07
	skillet / 0.19	pan / 0.16	scp. / 0.07	board / 0.06
salt	water / 0.40	bowl / 0.17	scp. / 0.05	food / 0.04
	water / 0.33	bowl / 0.14	diet / 0.06	scp. / 0.04
dish	table / 0.30	oven / 0.24	menu / 0.11	pan / 0.03
	table / 0.20	oven / 0.16	hand / 0.12	top / 0.04

we show the effect of more helpful entity specification.

2) *Tool Inference*: A similar approach that also includes the action domain ontology uses the preposition *with* to infer relations between actions and tools. The following syntactic pattern matches a verb *#action* and a noun *#tool* from the respective domain ontology, linked together by *with*:

$$((\#action, verb), (with, prep), (\#tool, noun)). \quad (9)$$

Table VI shows the three most probable tools for different actions from the kitchen domain. The results are shown for both the manually created domain ontology (upper rows, highlighted) and the automatically learnt one (lower rows).

3) *Class Inference*: Another possible result the system can compute is the probability that a word falls into a certain class of syntactic pattern. For example, given the above pattern (3), the system can approximate the probability that a word names an object or a location (Table VII). These results

TABLE VI: Results for tool inference

Highlighted rows: Manually created domain ontology. Non-highlighted rows: Automatically learnt domain ontology.

object	first	second	third
cut	knife / 0.80	fork / 0.01	machine / 0.01
	knife / 0.68	hand / 0.04	world / 0.03
flip	spatula / 0.89	spoon / 0.06	fork / 0.03
	spatula / 0.65	hand / 0.24	spoon / 0.05
mash	fork / 0.58	spoon / 0.16	butter / 0.09
	fork / 0.59	spoon / 0.16	butter / 0.09
stir	spoon / 0.50	fork / 0.20	spatula / 0.08
	spoon / 0.48	fork / 0.19	spatula / 0.08

can be used to improve the location inference results, e.g., by dropping words that seem unlikely to name a location.

TABLE VII: Results for class inference

Manually created domain ontology (left), automatically learnt domain ontology (right)

symbol	object	location	object	location
wine	0.80	0.20	0.85	0.15
water	0.48	0.52	0.56	0.44
meat	0.77	0.23	0.81	0.19
bowl	0.11	0.89	0.16	0.84
sugar	0.93	0.07	0.95	0.05
mixture	0.72	0.28	0.77	0.23
pan	0.17	0.83	0.18	0.82
oil	0.82	0.18	0.83	0.17
oven	0.12	0.88	0.11	0.89
salt	0.95	0.05	0.96	0.04

4) *Computation Time*: In this work, the domain ontology learning and distribution computation steps are considered to be run offline. However we note that the computation time for these steps depends heavily on the sizes and representations of \mathcal{C}_D and \mathcal{C}_I . Processing the Google Corpus⁸ requires especially high computational power. On the other hand, the inference step consists of simple lookups in precomputed distributions, and can therefore be done online.

V. PLANNING WITH COMMON SENSE KNOWLEDGE

In this section we show how the domain knowledge induced by the processes described above can be used with an automated planning system to improve the quality of generated plans. We have chosen to use the PKS (*Planning with Knowledge and Sensing* [19], [20]) planner for this task, since PKS has previously been deployed in robot environments like the one in Fig. 1 [21]. However, one of the strengths of the above approach is that it is not planner (or domain) dependent, and the method we outline for PKS can be adapted to a range of different planners and domains.

As an example scenario, we will focus on the use of spatial relations in a small kitchen domain. The domain contains the entities *cereal*, *counter*, *cup*, *cupboard*, *juice*, *plate*, *refrigerator* and *stove*. Table VIII shows the results of the location inference method. We will first postprocess this data for planning by considering the entity *juice*.

A. Postprocessing

Given the initial domain of objects, and using pattern (3), we can approximate the probability of an object o being

⁸The Google Arcs Corpus contains 38G of compressed text.

TABLE VIII: Location inference for a small domain

Omitted values are zero.

object	counter	cup	cupboard	dishwasher	juice	plate	refrigerator	stove
cereal			1.00					
cup	0.25	0.33	0.13	0.01		0.18	0.02	0.08
juice	0.02	0.43			0.21	0.08	0.26	
plate	0.14	0.10	0.07			0.58	0.06	0.06

spatially related to a location l by issuing a relation query:

$$P(\text{loc} = l | \text{obj} = o) = \rho\left(\{(obj, o), (loc, l)\}, \text{loc}\right). \quad (10)$$

To put these results into a suitable form for planning, we introduce the predicate *at* and output the computed likelihoods for pairs of objects. The resulting relations that are extracted, and their likelihoods, are shown in the top half of Table IX.

The postprocessor must now refine the results, possibly making use of additional information about the structure of the planning domain and the types of objects that are available. Refinement can be done in three possible ways:

- 1) *Symbol Mapping*: A word that describes an object in natural language may not necessarily match the symbol name for that object in the planning domain. This is currently corrected by an appropriate mapping process that uses a dictionary of likely synonyms. E.g., the word *refrigerator* may be mapped to *fridge*.
- 2) *Type Filtering*: Many planners have the concept of object *types*, which enables us to filter relations that have entity arguments of the incorrect type. Assuming the planning domain provides us with a type *location* that is required for the second argument of *at*, the postprocessor can then remove the extracted relations *at(juice, cup)*, *at(juice, juice)*, and *at(juice, plate)*, since the entities *cup*, *juice*, and *plate* are not locations.
- 3) *Instantiation*: The symbols extracted by our processes will often refer to *classes* of objects, rather than the specific object identifiers used by the planner. Making use of type information in the planning domain, the postprocessor can instantiate objects of the appropriate types from the extracted relational information. For instance, the class *juice* might be instantiated into two objects, *applejuice* and *orangejuice*. These objects can subsequently be substituted in any relations that contains the appropriate class type.

The final set of postprocessed relations from our example is shown in the bottom half of Table IX. We note that the necessity and possibility of applying these postprocessing steps depends on the nature of the planning domain. Furthermore, the information that is needed to perform the postprocessing, i.e., the symbol mapping table or the type information, needs to be manually encoded in the planning domain.

Given the postprocessed set of relations, the final step is to decide how this information will be included in the planning domain. For planners that work with probabilistic representations, the relation/likelihood information could be

TABLE IX: Extracted and postprocessed relations

Extracted	
Relations	Likelihood
<i>at(juice, cup)</i>	0.43
<i>at(juice, refrigerator)</i>	0.27
<i>at(juice, juice)</i>	0.21
<i>at(juice, plate)</i>	0.08
<i>at(juice, counter)</i>	0.02
Postprocessed	
Relations	Likelihood
<i>at(applejuice, fridge)</i>	0.27
<i>at(applejuice, counter)</i>	0.02
<i>at(orangejuice, fridge)</i>	0.27
<i>at(orangejuice, counter)</i>	0.02

directly encoded. For planners like PKS that do not deal with probabilities, there are two main possibilities:

- 1) The most probable location for each object could be encoded as a single fact in the planner’s knowledge, i.e., *at(applejuice, fridge)* and *at(orangejuice, fridge)*.
- 2) Some or all of the most probable locations could be encoded as a disjunction of possible alternatives, i.e., *at(applejuice, fridge) | at(applejuice, counter)*, and *at(orangejuice, fridge) | at(orangejuice, counter)*.

Depending on the domain, either form may be appropriate.

B. Plan Generation

Consider the task of finding the apple juice container in the kitchen. In the absence of precise information as to the object’s location, but knowing there are various places in the kitchen where objects could be located (e.g., *counter*, *cupboard*, *fridge*, *stove*), a planner could potentially build a plan for a robot to exhaustively check all locations: *move-robot-to-counter, check-for-apple-juice, if not present move-robot-to-cupboard, check-for-apple-juice, if not present move-robot-to-fridge*, etc., until all locations have been checked. If the robot does not have information-gathering capabilities to check for the apple juice in a particular location, the planner may not be able to generate such a plan at all.

With the availability of more certain information about the location of the apple juice, the planner can potentially eliminate some parts of the plan (e.g., by ignoring certain locations), or at least prioritise certain likely locations over others, resulting in higher quality plans. For instance, in the case that the planner had the knowledge *at(applejuice, fridge)*, resulting from the above relation extraction process, then the planner could build the simple plan *move-robot-to-fridge*, under the assumption that the extracted information was true.

Similarly, if the planner had the disjunctive information *at(applejuice, fridge) | at(applejuice, counter)* then the planner could build the plan: *move-robot-to-fridge, check-for-apple-juice, if not present move-robot-to-counter*. Again, this plan improves on the exhaustive search plan by only considering the most likely locations for the apple juice, resulting from the extracted relational information.

One inherent danger when dealing with common sense knowledge is that the plans that are built from such information alone may ultimately fail to achieve their goals in

the real world. For instance, even though relation extraction provides us with likely locations for the apple juice, there is no guarantee that this is the way the robot's world is actually configured. (E.g., another robot may have left the apple juice on the stove.) However, such information does give us a starting point for building plans, in the absence of more certain information, and can also aid plan execution monitoring to guide replanning activities in the case of plan failure. (E.g., if a plan built using common sense knowledge fails to locate the apple juice, fall back to the exhaustive search plan for the locations that haven't been checked.)

Finally, we note that the use of common sense knowledge may improve the efficiency of plan generation, since in general more specific information helps constrain the plan generation process. However, plan generation time is both domain and planner dependent, and it is difficult to quantify any improvements without experimentation. (E.g., planning time went from 0.003s to 0.001s in our small examples.)

VI. CONCLUSION AND FUTURE WORK

We have presented two techniques for reducing the amount of prior, hardcoded knowledge that is necessary for building a robotic planning domain. Using the methods described here, a domain ontology of object and action types can be defined automatically, over which user-defined relations can be inferred automatically from sources of natural language text. The resulting representation of common sense domain knowledge has been tested using an automated planning system, improving the quality of the generated plans.

As future work, we are exploring a number of improvements to our techniques. First, more specialized corpora C_I , longer syntactic patterns, or databases of common sense knowledge might help in overcoming the sparsity of common sense information in text sources. Second, the location inference does not perform any checks for plausibility. While the class inference will help in filtering results that are not locations at all, additional methods are needed to differentiate between locations for temporary storage and locations for long-term storage. Another interesting improvement would be the generalization of inferred relations to still missing knowledge. For example one could conclude by analyzing text sources that *bowl* and *dish* are conceptually similar and therefore apply relations inferred for *bowls* also to *dishes*. Finally, we are investigating the application of our methods to robot domains other than the kitchen environment.

ACKNOWLEDGMENT

The research leading to these results received funding from the European Union's 7th Framework Programme FP7/2007-2013, under grant agreement N^o270273 (Xperience).

REFERENCES

- [1] M. Terno, U. Klank, D. Pangercic, and M. Beetz, "Web-enabled robots," *Robotics & Automation Magazine*, vol. 18, no. 2, pp. 58–68, 2011.
- [2] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "Roboearth," *Robotics Automation Magazine, IEEE*, vol. 18, no. 2, pp. 69–82, 2011.
- [3] K. Welke, P. Kaiser, A. Kozlov, N. Adermann, T. Asfour, M. Lewis, and M. Steedman, "Grounded spatial symbols for task planning based on experience," in *13th International Conference on Humanoid Robots (Humanoids)*. IEEE/RAS, 2013.
- [4] A. Kasper, R. Becher, P. Steinhaus, and R. Dillmann, "Developing and analyzing intuitive modes for interactive object modeling," in *ICMI '07: Proceedings of the 9th international conference on Multimodal interfaces*. New York, NY, USA: ACM, 2007, pp. 74–81.
- [5] T. Asfour, K. Regenstein, P. Azad, J. Schröder, N. Vahrenkamp, and R. Dillmann, "ARMAR-III: An integrated humanoid platform for sensory-motor control," in *IEEE International Conference on Humanoid Robots (Humanoids)*, 2006, pp. 169–175.
- [6] T. Asfour, P. Azad, N. Vahrenkamp, K. Regenstein, A. Bierbaum, K. Welke, J. Schröder, and R. Dillmann, "Toward humanoid manipulation in human-centred environments," *Robotics and Autonomous Systems*, vol. 56, no. 1, pp. 54–65, 2008.
- [7] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation," in *Proceedings of the 25th National Conference on Artificial Intelligence*. AAAI, 2011, pp. 1507–1514.
- [8] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Toward understanding natural language directions," in *Proceedings of the 5th International Conference on Human-Robot Interaction (HRI)*. IEEE, 2010, pp. 259–266.
- [9] D. Chen and R. Mooney, "Learning to interpret natural language navigation instructions from observations," in *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-2011)*, 2011, pp. 859–865.
- [10] P. Singh, T. Lin, E. T. Mueller, G. Lim, T. Perkins, and W. L. Zhu, "Open mind common sense: Knowledge acquisition from the general public," in *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*. Springer, 2002, pp. 1223–1237.
- [11] C. Teo, Y. Yang, H. Daumé III, C. Fermüller, and Y. Aloimonos, "A corpus-guided framework for robotic visual perception," in *Workshop on Language-Action Tools for Cognitive Artificial Agents, held at the 25th National Conference on Artificial Intelligence*. San Francisco: AAAI, 2011, pp. 36–42.
- [12] —, "Toward a Watson that sees: Language-guided action recognition for robots," in *IEEE International Conference on Robotics and Automation*. St. Paul, MN: IEEE, 2012, pp. 374–381.
- [13] M. Tamosiunaite, I. Markelic, T. Kulvicius, and F. Wörgötter, "Generalizing objects by analyzing language," in *11th International Conference on Humanoid Robots (Humanoids)*. IEEE/RAS, 2011, pp. 557–563.
- [14] K. Zhou, M. Zillich, H. Zender, and M. Vincze, "Web mining driven object locality knowledge acquisition for efficient robot behavior," in *2012 International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ, 2012, pp. 3962–3969.
- [15] D. Klein and C. D. Manning, "Accurate unlexicalized parsing," *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics ACL 03*, vol. 1, pp. 423–430, 2003.
- [16] G. A. Miller, "WordNet: a lexical database for English," *Communications of the ACM*, vol. 38, pp. 39–41, 1995.
- [17] C. Fellbaum, *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press, 1998.
- [18] Y. Goldberg and J. Orwant, "A dataset of syntactic-ngrams over time from a very large corpus of english books," in *Second Joint Conference on Lexical and Computational Semantics*, 2013, pp. 241–247.
- [19] R. P. A. Petrick and F. Bacchus, "A knowledge-based approach to planning with incomplete information and sensing," in *International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2002)*, 2002, pp. 212–221.
- [20] —, "Extending the knowledge-based approach to planning with incomplete information and sensing," in *International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 2004, pp. 2–11.
- [21] R. Petrick, N. Adermann, T. Asfour, M. Steedman, and R. Dillmann, "Connecting knowledge-level planning and task execution on a humanoid robot using Object-Action Complexes," in *Proceedings of the International Conference on Cognitive Systems (CogSys 2010)*, 2010.