SHAPELURN: An Interactive Language Learning Game with Logical Inference

Katharina Stein **Leonie Harter** **Luisa Geiger**
Department of Language Science and Technology
Saarland Informatics Campus
Saarland University, Germany
{kstein, leonieh, lgeiger}@coli.uni-saarland.de

Abstract

We investigate if a model can learn natural language with minimal linguistic input through interaction. Addressing this question, we design and implement an interactive language learning game that learns logical semantic representations compositionally. Our game allows us to explore the benefits of logical inference for natural language learning. Evaluation shows that the model can accurately narrow down potential logical representations for words over the course of the game, suggesting that our model is able to learn lexical mappings from scratch successfully.

1 Introduction

An open question in NLP research is how models can learn natural language most successfully and effectively. Many state-of-the-art semantic parsers and machine learning algorithms are dependent on large data sets for successful training. This poses a problem when using NLP applications for low-resource languages or specific domains for which only little annotated training data is available if any at all (Klie et al., 2020). *Interactive NLP Systems* can overcome these problems as they start with a small or even empty set of training data that gets extended based on user feedback for the predictions the model makes based on its current parameters (Lee et al., 2020). Therefore, learning mappings from natural language to formal representations through interaction with a user is an attractive approach for low-resource settings. The model parameters are optimized based on feedback and the resulting data itself can be used as training data for other models avoiding costly manual annotations.

However, the interaction with a not yet fully trained model can get monotonous or can lead to frustration on the part of the users if they do not benefit from the interaction themselves (Lee et al., 2020). Wang et al. (2016) present an interactive language learning setting, called SHRDLURN, in which a model learns a language by interacting with a player in a game environment, hence making the interactive learning setting more attractive and fun for users. Their model is language independent and can be taught any language from scratch.

Here we follow Wang et al. (2016) and design and implement an interactive language learning game where a model learns to map natural language to logical forms in a compositional way based on the feedback provided by the player[1]. Whereas Wang et al. (2016)'s model learns to map instructions to executable logical forms, we aim to learn logical formulas that evaluate to truth values with respect to the current state of the game environment. This decision was taken because the additional information about the truth can be incorporated in the parsing and learning process in order to already restrict the potential logical formulas. Overall, we are trying to answer the following research questions: Can we implement a model that 1) can learn a natural language from scratch only from interacting with a user and 2) is not dependent on any language specific syntax and is hence language independent.

2 Previous Work

Several approaches map natural language to logical form for its ability to model inferences. Zettlemoyer and Collins (2005) present a learning algorithm for mapping sentences to their lambda calculus semantic representations and automatically inducing a combinatory categorical grammar (CCG). Zettlemoyer and Collins (2007) extend this algorithm to make the grammar more flexible. Pasupat and Liang (2015) also present a flexible semantic parsing framework, the floating parser, for learning mappings from natural language to logical forms

[1]The complete code is available under https://github.com/itsLuisa/SHAPELURN

Proceedings of the First Workshop on Interactive Learning for Natural Language Processing , pages 16–24

in the lambda dependency-based compositional semantic language (Liang, 2013). Liang and Potts (2015) present a framework for learning to map natural language utterances to logical forms that combines the principle of compositionality with a standard machine learning algorithm.

Current approaches that aim at overcoming the need for costful annotated training data include the interactive *human-in-the-loop* method where a human corrects annotations predicted by a machine learning system and this feedback is used to improve future predictions. Klie et al. (2020) use this approach for the task of Entity Linking and He et al. (2016) apply the approach to a CCG parser, thereby improving parsing performance. Goldwasser and Roth (2014) present a learning approach where a model learns to map natural language instructions to logical representations of solitaire game rules based on feedback. Finally, Zhang et al. (2018) present a game for grounded language acquisition where a human teaches an agent language from scratch in a natural language conversation.

3 From SHRDLURN to SHAPELURN

3.1 Game Design

Wang et al. (2016) designed their model to perform a block building task in 3-D space using natural language instructions from the player. The computer and player work together towards a shared goal (a specific block position) while only the player knows the goal and only the computer can move the blocks. The more successful the model learns the human's language, the faster this shared goal can be reached (Wang et al., 2016).

Based on this idea, we design a 2-D game environment where the model and player work towards teaching the model a language with the user giving descriptive input about the environment. The user is presented with a randomly generated picture displaying varying numbers of objects which have one of three shapes (circle, square, triangle) and one of four colors (red, blue, yellow, green) (see Figure 1). The picture corresponds to a 4×4-grid which is internally represented as a matrix allowing for simplified spatial calculations.

The user is asked to describe one or more of the displayed objects by typing in one phrase in a language of their choice. Importantly, this language should be kept consistent in order for the computer to recognize language specific patterns. The program proceeds by parsing the input into a logical formula, comparing it to the matrix and then making a guess on which object(s) the user was referring to by marking them with a black border. The user can then provide feedback in terms of selecting the right marking by skipping through the computer's guesses which show up in descending order according to their probability (see A.1). Like this, the feedback is specific enough for the model to learn lexical mappings.

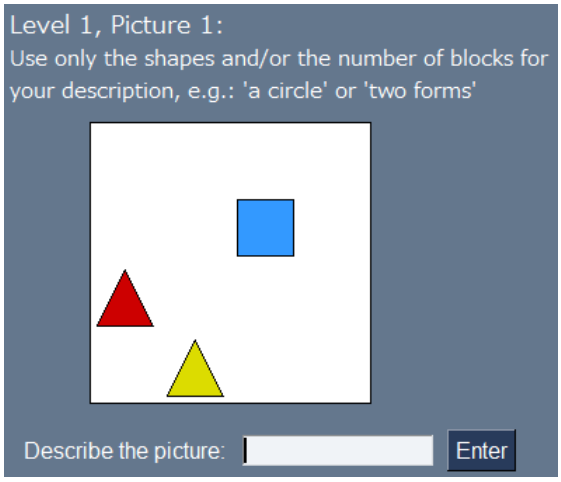

Figure 1: The interface displaying three randomly generated objects the user can use to build an utterance

This process is repeated over 4 levels of alternating complexity (regarding both the number of displayed blocks and the length of the input) each consisting of 20 (level 2) or 15 (other levels) pictures. The learning algorithm is responsible for the guesses to improve as the game proceeds and lets the model adapt to the input language.

3.2 Preprocessing and Parsing

We tokenize, lowercase and stem the input, since learning is much quicker if it is clear that e.g. *triangle* and *triangles* refer to the same shape. In order to stem language independently, we use a cosine similarity based heuristic. To compare two tokens, we transform them into vectors with length of the longer one. If they have the same character at a position, the vectors get a 1 there. Otherwise one vector gets a 1 and the other a -1. If the cosine similarity of the vectors is > 0.65, we assume the tokens to belong to the same word (see Figure 2).

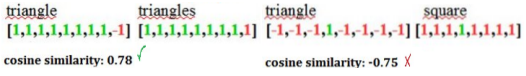

Figure 2: Two stemming examples

Since Liang and Potts (2015)'s framework[2], which we use as groundwork, employs a CYK parser that forces players to adhere to a strict syntax, we equipped it with Pasupat and Liang (2015)'s floating parser instead. This parser stores intermediate results in a chart according to their semantic category c and size s (Figure 3), but does not consider which indices the covered tokens span. This allows to parse syntactic structures without binding the user to a certain syntax. We adjust the three derivation rules for parsing to our grammar:

(1) $(TokenSpan, i, j)[s] \rightarrow (c, 1)[f]$

(2) $\emptyset \rightarrow (c, 1)[f]$

(3) $(c_1, s_1)[f_1] + (c_2, s_2)[f_2] \rightarrow (c, s_1 + s_2)[f_1(f_2)]$

Rule (1) is a lexical rule matching a token span from index i-j to a rule in the lexicon entry of the word in this span telling us which category c and which function f to use. Rule (2) allows us to establish lexical logical forms matching words we have not seen in the input and proceeds with these "imaginary tokens" as in (1). It is used to create the formula in the parse chart field (E,1) in Figure 3. Rule (3) combines parse items of categories the grammar allows to combine. We only allow each token to be used once per parse.

Previous work used beam search to address the huge number of possible parse trees (Wang et al., 2016; Pasupat and Liang, 2015). However, as beam search does not guarantee that the correct parse is kept we decided to not use a heuristic approach for pruning. Instead, we restrict the number of parses by building only formulas up to the size corresponding to the number of tokens in the input plus four. Additionally, we allow each token to be mapped to only one lexical rule per parse. This averts building non-sense constructions like *(exist([2])(blue(BF(triangle,all))))(exist([1])(blue(BF(square,all))))* for the utterance "two triangles and one blue square" (considering the picture in Figure 1).

	E	N	C	B	EN	BC	V
1	[exist]	[[1]]	[blue]	[BF(circle,all)]			
2					[exist([1])]	[blue(BF(circle,all))]	
4							[exist([1])(blue(BF(circle,all)))]

Figure 3: Parse Chart for "one blue circle"

3.3 Grammar

For our grammar we use the same overall structure as Liang and Potts (2015) (see A.3 for the complete grammar). The main information is encoded in the lexicon which is a dictionary that pairs words with a list of corresponding lexical rules. A lexical rule is a triple of a category (B, C, E, N, POS or CONJ), a logical form and a weight. For example, the word "red" is paired with (C, *red*, w), where C is the category, *red* the logical form as defined by the grammar and w the current weight.

The logical formulas for entries of category B and N are evaluated directly whereas the other logical forms are functions whose evaluation is specified separately using lambda calculus. For example, *red* is defined as the function $\lambda x(BF(red, x))$ where $BF(condition, list)$ is a function that yields all blocks from *list* that fulfill *condition*.

We use a set of binary CFG rules to define which categories can be combined and how logical formulas are applied to each other to yield larger formulas, e.g. BC $\rightarrow$ C B specifies that formulas of category C and B can be combined to a formula of category BC by applying C to B.

Each completed formula V is composed of at least one sub-formula of the categories B, N, E and C, and the categories POS and CONJ allow to build more complex formulas for inputs including relative positions and conjunctions.[3] Descriptions not specifying a color are handled by rule (2) of the floating parser that introduces a lexical rule of category C with an empty condition into the parse. For lower parsing complexity, users are instructed to mention only the objects, e.g. "a circle" instead of "there is a circle". We model the implicit existential quantification with a lexical rule for *exists* that gets introduced into each formula by rule (2).

3.4 Learning Algorithm

Like Wang et al. (2016), we aim to learn correct lexical rule(s) for all words in the lexicon. Initially, every new word gets paired with every lexical rule. Following Zettlemoyer and Collins (2005)'s approach in a simplified way, we delete the most unlikely pairs during training leaving the correct ones remain. We use Liang and Potts (2015)'s learning algorithm, a linear regression model optimized with stochastic gradient descent (SGD), which returns weight changes for word-rule pairs improving the model. Whereas Wang et al. (2016)'s features consist of n-grams and skip-grams for the utterance, tree-grams for the formulas and a formula depth,

[2]https://github.com/cgpotts/annualreview-complearning

[3]The denotation of a formula V consists of the truth of the description w.r.t. the picture and the list of blocks that make the description *true*.

our features only contain a list of word-rule pairs. This is sufficient, since the formula's structure and depth and the distances between combinable words are handled by the floating parser.

After a training round we get weight changes for all words in the input paired with the rules used to build the gold standard logical formulas and the ones predicted by the model. We sum up the weights for each pair after every training step. If a weight sum reaches the lower threshold of -0.1, we delete this rule for the corresponding word. If all pairs get weight change 0, SGD has converged, so the model is optimal for the current training batch. Hence, the word rule pairs used to build the formulas for the current training utterance must be the correct ones and all others can be deleted. If a weight sum reaches the upper threshold of 1.0, we assume this rule to be correct and delete all other rules for the word with weight sum ≤ 0.

As different formulas can have the same denotation (see Figure 4), we group all formulas evaluating to *true* by the guessed blocks they evaluate to. Only these guesses are then presented to the player. The formulas leading to the correct blocks, as determined by the user feedback, are used as gold standard training batch. During training we collate all possible parses. Otherwise too much information is lost, which causes deletions of correct rules. Liang and Potts (2015)'s cost function gave us too few weight changes $\neq 0$. Therefore, we average over all rules of a formula if this rule was also used in the gold formulas (value 1) or not (value 0):

$$\frac{1}{n}\sum_{i=1}^{n}\begin{cases}0 & if\,(w_i, r_i)\,in\,gold\,word\,rule\,pairs\\ 1 & otherwise\end{cases}$$

***n**: number of tokens* $\mathbf{w_i}$*: token number i*
$\mathbf{r_i}$*: rule applied to* w_i *to get current formula*
***gold word rule pairs**: word rule pairs leading to gold formula*

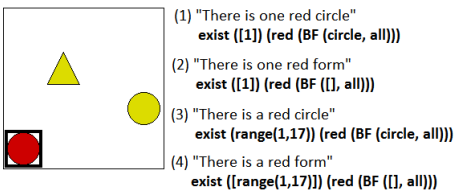

Figure 4: Four formulas with the same denotation

4 Evaluation

For a preliminary evaluation of the performance of the model we collected and analyzed data of seven participants who played the game in English (3), Spanish (1), German (2) and Esperanto (1). One of the participants completed two levels, four three levels and only two completed all four levels.

We planned to follow Wang et al. (2016) and count how often the user needs to click "NEXT" i.e. how far down the ranked parses the correct solution is. For our project to be viewed as successful, this number should decrease over the course of the game. Due to the simple game design, the exclusion of formulas evaluating to *false* and the grouping of identical markings, the total number possible markings is very low throughout the whole game and so is the number of clicks needed to arrive at the desired one ($M = 1.27$). But because of our simplified game setting, this cannot be directly compared to Wang et al. (2016)'s results. Since the number of clicks almost does not change in our case, it is not a meaningful measure for evaluating the improvement of the model.

To assess performance within the model itself, we analyze the remaining rules for each word. Initially, there are 20 possible rules per word as each new word gets paired with each rule from the lexicon and ideally exactly the correct rule(s) for each word should be left finally. Figure 5 shows the average number of rules per word left after each level. As our data set is very small the results have to be taken with a grain of salt. Nevertheless, the plot reveals that the model was able to decrease the number of possible rules per word by about 15 (see A.2). Manual investigation of the remaining rules at the end of level 3 revealed a total of 1202 deletions (63.9%), out of which only seven were falsely deleted (0.58%). This indicates that our model is able to successfully exclude incorrect mappings.

5 Discussion

Although, the setting of our language learning game is simpler than SHRDLURN, the 2-D grid environment allows to elicit different kinds of inputs that involve the composition of colors and shapes as well as relative spatial relations between objects that can be nested. In contrast to Wang et al. (2016), the player task in our game is not to give instructions to reach a specified goal state from the current state but to describe some part of the current state of the game (picture). Although

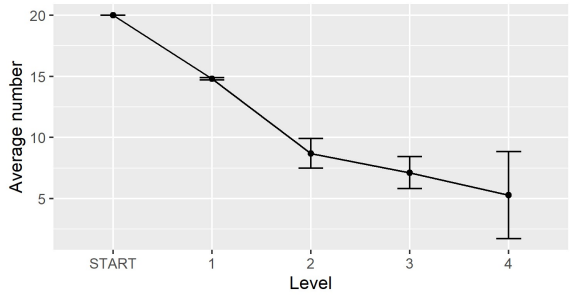

Figure 5: Mean number of rules per word left after each level. Error bars indicate 95% confidence intervals.

we restrict the complexity of the descriptions in the first levels to improve the first learning phase, the player task is in general a very open task as there are many ways to describe objects in a grid and the player can freely choose the objects to describe as well as the properties used to reference them. This makes the setting particularly interactive and allows to investigate in which ways humans choose and formulate their descriptions.

The design of the player task allows us to use the truth values of the parses in order to present only the *true* guesses to the player. Hence, the number of potential denotations is limited by the number of possibilities to mark different combinations of objects in the picture. This decreases the number of denotations the player can choose from compared to Wang et al. (2016) where the number of potential successor states for a current state is much higher. Although our design inhibits to use the number of clicks during the game as evaluation measurement, we see the overall low number of guesses as an advantage: the player spends less time on clicking through wrong guesses even in cases where the correct denotation is ranked very low what can improve the playing and success experience.

We find that the informativeness of feedback plays a crucial role in interactive learning. Our results indicate that making the current "knowledge" of the computer as explicit as possible, e.g. by marking all blocks mentioned in the input, could be a promising starting point as simple user feedback can provide enough information for learning.

During testing we found that the learning progress depends on specific combinations of descriptions and pictures. Learning appears to benefit from situations where the correct formula for the description differs in one lexical rule from other *true* parses: "a red circle" is more informative with respect to the meaning of "red" for a picture that shows a red circle and a circle in another color than for a picture displaying only one (red) circle.

The main advantage of using only input from the player to train the model is its independence of the availability of (annotated) training data as opposed to approaches requiring large data sets such as neural approaches. Hence, our approach is applicable for low resource settings. However, our model requires a grammar that covers all semantic concepts that can be part of the interaction. Due to our game design, the number of concepts our grammar needs to address is very limited but extending the domain requires increasing the number of handwritten rules. Therefore, scaling the model to larger domains would require the costful construction of a large-scale grammar.

Concerning the scalability of the game environment, the model could be easily adjusted to create more complex pictures as long as an adequate internal representation is found.

A key challenge of our work was the high uncertainty of the model. The computer has no initial knowledge about language and must consider each lexicon entry for each word. Additionally, the floating parser can combine the corresponding logical formulas in any order, discard tokens from the input and add additional logical forms. The number of parses is thus huge for short sentences and grows exponentially with sentence length, vastly increasing parsing and learning times. Future work could use beam search at higher levels to handle this.

6 Conclusion

We implement an interactive language learning game where the computer learns natural language based on user feedback. We find that learning interactively from scratch with a language independent model is complex due to the huge number of potential parses. Our results indicate that our model is able to learn language through interaction and low-resource domains and languages could benefit from such an approach. Future work will address the trade-off between increasing flexibility and increasing processing times.

Acknowledgments

We would like to thank Dr. Lucia Donatelli for the valuable discussions and support throughout the project development and writing process. Further, we would like to express our gratitude towards the participants of our evaluation experiment.

References

Dan Goldwasser and Dan Roth. 2014. Learning from natural instructions. *Machine learning*, 94(2):205–232.

Luheng He, Julian Michael, Mike Lewis, and Luke Zettlemoyer. 2016. Human-in-the-loop parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2337–2342, Austin, Texas. Association for Computational Linguistics.

Jan-Christoph Klie, Richard Eckart de Castilho, and Iryna Gurevych. 2020. From Zero to Hero: Human-In-The-Loop Entity Linking in Low Resource Domains. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6982–6993, Online. Association for Computational Linguistics.

Ji-Ung Lee, Christian M. Meyer, and Iryna Gurevych. 2020. Empowering Active Learning to Jointly Optimize System and User Demands. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4233–4247, Online. Association for Computational Linguistics.

Percy Liang. 2013. Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*.

Percy Liang and Christopher Potts. 2015. Bringing machine learning and compositional semantics together. *Annual Review of Linguistics*, 1(1):355–376.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Sida I. Wang, Percy Liang, and Christopher D. Manning. 2016. Learning language games through interaction. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2368–2378, Berlin, Germany. Association for Computational Linguistics.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 658–666, Arlington, Virginia, USA. AUAI Press.

Luke S. Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687.

Haichao Zhang, Haonan Yu, and Wei Xu. 2018. Interactive language acquisition with one-shot visual concept learning through a conversational game. *arXiv preprint arXiv:1805.00462*.

A Appendices

A.1 Game Design

Level	Instruction
0	Welcome to SHAPELURN, where you can teach the computer any language of your choice! You will be looking at different pictures and describing them to the computer in one sentence. There will be four levels with different constraints on the descriptions. Please use short sentences in the first two levels and do not use negation at all.
1	Use only the shapes and/or the number of blocks for your description e.g.: *'a circle'* or *'two forms'*
2	You can additionally describe the blocks by color e.g: *'two blue forms'*
3	Now you can describe relations between blocks and use conjunction (please don't use colors) e.g.: *'a circle under a square'*
4	Describe whatever you want!

Table 1: The overall instructions for the input descriptions (0) and the level specific constraints for the descriptions.

Figure 6 - 8 illustrate the course of the game for an example picture and the input description "two triangles". The user is shown a picture and enters a description. Then the computer makes a guess and the user clicks NEXT until the correct guess is shown.

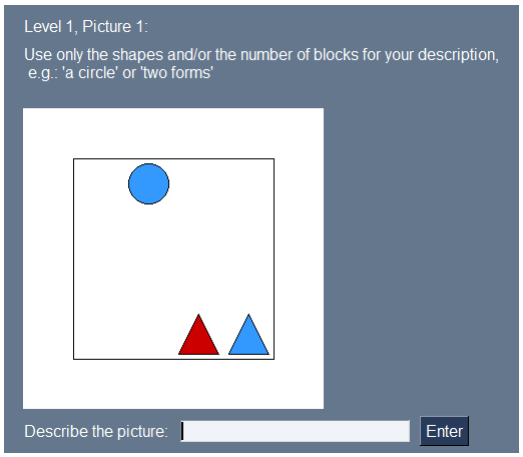

Figure 6: Grid generated by the model in level 1

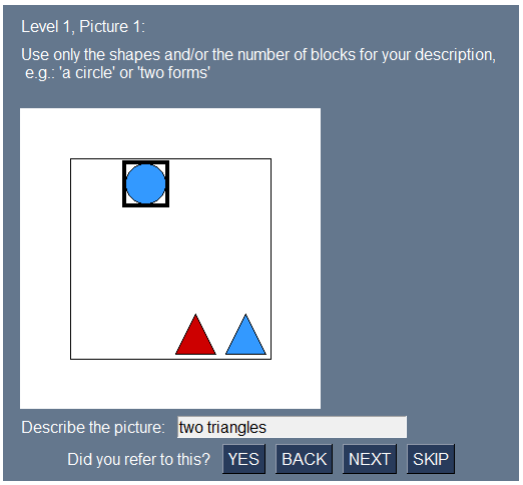

Figure 7: First guess of the model, user clicks NEXT

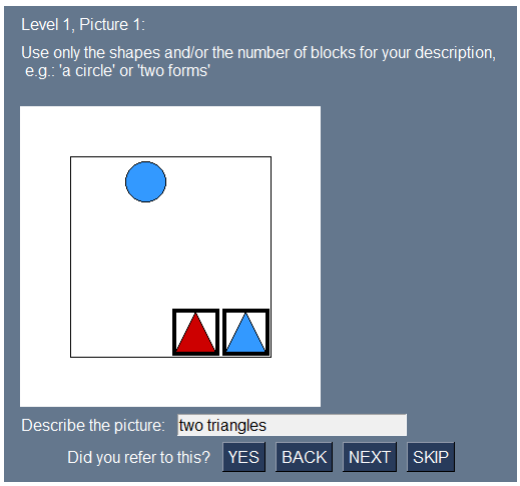

Figure 8: Next guess is correct, user clicks YES

A.2 Evaluation

Level	Mean	SD	Participants
1	14.801	0.114	7
2	8.691	1.319	7
3	7.116	1.250	6
4	5.280	0.396	2

Table 2: Mean and sd for the average number of rules per word in the lexicon at the end of each level

A.3 The Grammar

	Rule
1	V $\rightarrow$ EN BC
2	V $\rightarrow$ CONJ_1 V
3	CONJ_1 $\rightarrow$ CONJ V
4	EN $\rightarrow$ E N
5	BC $\rightarrow$ C B
6	BC $\rightarrow$ POS_NB BC
7	POS_NB $\rightarrow$ POS_N BC
8	POS_N $\rightarrow$ POS N

Table 3: The rules of the CFG grammar used to derive the input utterances and the logical forms

	Lexical Rule: (category, logical form, weight)	**Example Word**
1	(B, $BF([\lambda b(b.shape == "rectangle")], all), w)$	"square"
2	(B, $BF([\lambda b(b.shape == "circle")], all), w)$	"circle"
3	(B, $BF([\lambda b(b.shape == "triangle")], all), w)$	"triangle"
4	(B, $BF([\quad], all), w)$	"form"
5	(N, $range(1, 17), w)$	"a"
6	(N, $[1], w)$	"one"
7	(N, $[2], w)$	"two"
8	(N, $[3], w)$	"three"
9	(C, $green, w)$	"green"
10	(C, $blue, w)$	"blue"
11	(C, $yellow, w)$	"yellow"
12	(C, $red, w)$	"red"
13	(POS, $over, w)$	"over"
14	(POS, $under, w)$	"under"
15	(POS, $next, w)$	"next [to]"
16	(POS, $left, w)$	"[to the] left [of]"
17	(POS, $right, w)$	"[to the] right [of]"
18	(CONJ, $und, w)$	"and"
19	(CONJ, $oder, w)$	"or"
20	(CONJ, $xoder, w)$	"or"
21	(C, $anycol, w)$	$\emptyset$
22	(E, $exist, w)$	$\emptyset$ / [there is]
Function $BF(condition, all)$ returns all blocks from the list of all blocks of the picture that fulfill condition $condition$		

Table 4: The lexicon of the grammar where each lexical rule is triple of (category, logical form, weight) and English example words for each rule.

	Logical Form from Lexicon	Function for interpretation
1	$exist$	$\lambda n(\lambda b(update\_guess(b)\ and\ len(b)\ in\ n))$
2	$green$	$\lambda x(BF([\lambda b(b.color == "green")],\ x))$
3	$blue$	$\lambda x(BF([\lambda b(b.color == "blue")],\ x))$
4	$yellow$	$\lambda x(BF([\lambda b(b.color == "yellow")],\ x))$
5	red	$\lambda x(BF([\lambda b(b.color == "red")],\ x))$
6	$anycol$	$\lambda x(BF([\ \ ],\ x))$
7	$over$	$\lambda n(\lambda x(\lambda y(PT(y,\ x,\ n,\ "o"))))$
8	$under$	$\lambda n(\lambda x(\lambda y(PT(y,\ x,\ n,\ "u"))))$
9	$next$	$\lambda n(\lambda x(\lambda y(PT(y,\ x,\ n,\ "n"))))$
10	$left$	$\lambda n(\lambda x(\lambda y(PT(y,\ x,\ n,\ "l"))))$
11	$right$	$\lambda n(\lambda x(\lambda y(PT(y,\ x,\ n,\ "r"))))$
12	und	$\lambda v1(\lambda v2(v1\ and\ v2))$
13	$oder$	$\lambda v1(\lambda v2(v1\ or\ v2))$
14	$xoder$	$\lambda v1(\lambda v2((v1\ and\ not\ v2)or(v2\ and\ not\ v1))))$
Function $BF(condition,\ x)$ returns all blocks from the list x that fulfill condition $condition$		
Function $PT(y,\ x,\ n,\ "pos")$ returns all blocks from the list y that stand in position "pos" to n blocks from list x		
Function $update\_guess(b)$ returns a list of all mentioned blocks by recursively backtracking from the blocks in list b		

Table 5: The functions used to interpret the logical forms for the categories E, C, POS and CONJ.

	CFG	Logical Form	Denotation
1	B → circle	*BF*(circle, all)	the list with all blocks of the picture with shape circle
2	N → one	[1]	[1]
3	C → blue	*blue*	the *C* s.t. *C*(x) returns a list of all blocks of x with color blue
4	POS → over	*over*	the *POS* s.t. *POS*(y)(x)(n) returns the sublist of blocks from y that are located over n blocks from x
5	E → ∅	*exist*	the *E* s.t. *E*(b)(n) returns True if length of b satisfies n and False otherwise
6	BC → C B	*C*(B)	application of denotation of *C* to denotation of B
7	V → EN BC	*EN*(BC)	application of denotation of *EN* to denotation of *BC*
8	EN → E N	*E*(n)	application of denotation of *E* to denotation of n
Input utterance: "one blue square over a red triangle" Logical Form: $exist([1])(over(range(1,17))(blue(BF(square,all)))(red(BF(triangle,all))))$ Denotation: True and the list of guessed blocks consisting of the blue square and the red triangle			

Table 6: Illustration of the way in which the grammar works for the example "[there is] one blue square over a red triangle". The upper part shows the lexical rules and the mid part the combination rules needed for the example sentence. The lower part shows the input utterance with its simplified logical form and the corresponding denotation with respect to the picture in Figure 1 in the paper.