

# Spatial Reasoning and Prism Implementation and Comparison

Master Project Report

**Ashwath Sampath**

MSc. Informatik

Matrikelnummer: 4368430

SS2018



Cognitive Computation Group  
Department of Informatik  
Albert-Ludwigs-Universität Freiburg  
17 August 2018

# Contents

<b>1</b>	<b>Goal of the Project</b>	<b>2</b>
<b>2</b>	<b>Introduction</b>	<b>2</b>
2.1	Verification vs. Generation Problems in Spatial Reasoning . . . . .	2
2.2	Determinate, Indeterminate and Inconsistent problems . . . . .	2
2.2.1	Determinate problems with a single solution . . . . .	3
2.2.2	Inconsistent problems . . . . .	3
2.2.3	Indeterminate problems with a single solution . . . . .	3
2.2.4	Indeterminate problems with multiple solutions . . . . .	3
2.3	Other types of problems . . . . .	4
2.3.1	Combination problems . . . . .	4
2.3.2	Generate all problems . . . . .	4
2.4	Mental Models . . . . .	4
<b>3</b>	<b>Theory and Implementation</b>	<b>5</b>
3.1	Spatial Parser . . . . .	5
3.1.1	Lexicon and Grammar . . . . .	6
3.1.2	Spatial Parser code . . . . .	6
3.2	Spatial Reasoning theory: Johnson-Laird/Byrne . . . . .	6
3.3	PRISM theory and implementation . . . . .	7
<b>4</b>	<b>Scenarios: according to Python implementations of both theories</b>	<b>8</b>
<b>5</b>	<b>Python Implementation and Execution</b>	<b>11</b>
5.1	Python Module Organization . . . . .	11
5.2	Types of sample problems . . . . .	12
5.3	Execution . . . . .	12
5.3.1	Classical spatial reasoning model (Johnson-Laird and Byrne) . . . . .	12
5.3.2	Prism Model . . . . .	12
5.4	Problems from Tables 1 and 2 in the Python program . . . . .	13
5.5	Problems from Prism paper [5] in Python program . . . . .	13

# 1 Goal of the Project

To compare two different models of spatial reasoning – Johnson-Laird and Byrne’s 1989 model and Ragni and Knauff’s 2013 PRISM model – after converting the former’s Lisp code into Python and enhancing it. The PRISM model is built independently, using the same framework (parser and high-level functions), but a different mechanism for getting alternative models.

## 2 Introduction

All of us reason spatially on a daily basis. Spatial reasoning is used to plan a route, to understand the positions of objects with respect to each other, and to locate objects. For example, let’s say we know that a bakery is to the left of a bank, and we are standing in a supermarket behind the bank. Now, as we know the relations between the three entities, we know the precise direction in which we need to walk (forward and left) to reach the bakery from the bank.

### 2.1 Verification vs. Generation Problems in Spatial Reasoning

Two different paradigms of spatial reasoning problems are specified in [5]: verification and generation. In verification problems, there are a group of premises and a conclusion which has to be verified. The question to answer in verification problems is: is the conclusion valid given the preceding premises? The following one-dimensional time series (problem 1) is an example of a verification problem.

#### Problem 1

*The square is on the right of the circle.  
The circle is on the right of the triangle.  
So, the square is on the right of the triangle.*

Here, the third premise is a conclusion premise, and for this problem, the conclusion is valid given the preceding premises.

On the other hand, generation problems are problems where the conclusion is not given, but is instead explicitly derived from the premises. The question to answer in generation problems is: what is the relation between two entities mentioned in the premises? Problem 2 shows a generation problem with the same entities mentioned in Problem 1.

#### Problem 2

*The square is on the right of the circle  
The circle is on the right of the triangle  
What is the relation between the square and the triangle?*

Here, the third premise is a question to be answered based on the knowledge gleaned from the previous premises. The answer, of course, is the conclusion premise in Problem 1.

### 2.2 Determinate, Indeterminate and Inconsistent problems

Spatial reasoning problems can also be categorized based on whether the conclusion is valid given the previous premises or not (inconsistent or not). Another thing to consider is whether multiple *models* can be built from a set of premises or if only a single model can be built. The word ‘models’ refers to mental models, which are graphical representations formed from the premises. Mental models are explained in more detail in Section 2.4.

### 2.2.1 Determinate problems with a single solution

Problem 1 is a very simple determinate verification problem, i.e., it has a single solution. Problem 2, on the other hand, is a determinate generation problem, i.e., one conclusion can be generated from one model.

### 2.2.2 Inconsistent problems

Inconsistent problems are verification problems in which the conclusion does not follow from the previous premises. It is clear from Problem 3 below that the third premise (the conclusion) is exactly opposite to the correct conclusion.

#### Problem 3

*The square is on the right of the circle.  
The circle is on the right of the triangle.  
So, the square is on the left of the triangle.*

### 2.2.3 Indeterminate problems with a single solution

These problems have a single solution, even though multiple *models* may be formed from the premises. Problems 4 and 5 show two such problems. Problem 4 is a verification problem, while problem 5 is a generation problem.

#### Problem 4

*A is on the right of B  
C is on the left of A  
D is in front of A  
E is in front of C  
So, D is on the right of E.*

#### Problem 5

*A is on the right of B  
C is on the left of A  
D is in front of A  
E is in front of C  
What is the relation between D and E?*

In both cases, the following 2 models are formed:

C B A	B C A
E D	E D

As is evident, only one relation exists between D and E in both models: D is on the right of E.

### 2.2.4 Indeterminate problems with multiple solutions

In these problems, there are different relations between two tokens in different models. Again, we can have either verification or generation problems. Consider the following problem, Problem 6. It is a verification problem where D can be to the right of E (as the conclusion premise says) or it can be the opposite: D is to the left of E. Therefore, no valid conclusion can be made in this problem.

Problem 7 is the generation version of the same problem, which has exactly the same result – no valid conclusion can be formed from the premises.

**Problem 6**

*A is on the right of B*  
*C is on the left of A*  
*D is in front of B*  
*E is in front of C*  
*So, D is on the right of E.*

**Problem 7**

*A is on the right of B*  
*C is on the left of A*  
*D is in front of B*  
*E is in front of C*  
*What is the relation between D and E?*

In both cases, the following 2 models are formed:

C B A	B C A
E D	D E

**2.3 Other types of problems****2.3.1 Combination problems**

These are problems considered in the Lisp code [3] associated with [1], but which do not play a role in the PRISM theory. Here, the third premise combines together tokens defined in the first two premises.

*A is on the left of B.*  
*C is on the right of D.*  
*A is in front of C*

**2.3.2 Generate all problems**

Another possibility defined in [5] is that reasoners may be asked to generate all possible models based on a set of premises. Problem 8 is the same as problem 7, except for the last premise, where all possible models are asked for. The result is the same: the reasoner draws the only 2 possible models.

**Problem 8**

*A is on the right of B*  
*C is on the left of A*  
*D is in front of B*  
*E is in front of C*  
*Generate all models.*

**2.4 Mental Models**

*"Mental models are psychological representations of real, hypothetical or imaginary situations",* according to the Princeton mental models website [4]. These models can represent both concrete and abstract concepts. According to the mental models theory, people use their understanding of the premises to build a model, in which the tokens represent physical objects (or concepts). This model, which can be an image, a spatial array or a scenario, is used by people as a basis for reasoning.

In the next section, we will look at the two theories of Spatial Reasoning mentioned in Section 1: Johnson-Laird/Byrne's Spatial Reasoning theory and Ragni/Knauff's PRISM theory.

### 3 Theory and Implementation

In this section, the two theories – the classical Spatial Reasoning theory ([1] and [2]) and the Prism theory ([5]) – are described briefly, along with the implementation details. Even though they are different theories, they are built on the same framework ([3]), of which the first part is the spatial parser.

#### 3.1 Spatial Parser

The spatial parser is used to produce an *intensional representation* of each premise in turn. A three-dimensional spatial array is used to model different kinds of relations such as ‘to the right of’, ‘behind’ and ‘above’. The origin is initially taken at 0,0,0 and the three axes correspond to left-right movement, back-front movement, and top-down movement. Fig. 2 shows a representation of the 9 possible relations that the parser takes into account while creating the intentional representation. ‘Beside’, ‘between’, ‘among’ and ‘next to’, all of which don’t have a specific directionality, are out of scope as far as this parser is concerned. Another important thing to note is that directions can be ambiguous. It can either be intrinsic (object’s

#### *The Lexical Semantics for a Set of Prepositions Expressing Spatial Relations*

Relation	Left-right	Back-front	Top-down
In the same place as	0	0	0
On the right of	1	0	0
On the left of	-1	0	0
In front of	0	1	0
In back of	0	-1	0
Behind	0	-1	0
Above	0	0	1
On top of	0	0	1
Below	0	0	-1

*Note.* 1 and -1 signify the directions in which to scan to meet the semantics of the relations. 0 means hold a value constant.

Figure 1: Lexical semantics

point of view) or deictic (speaker’s point of view). The parser specifies directions in the deictic sense. So, ‘in front of’ has the semantics (0 1 0), while behind has the semantics (0 -1 0).

The parser finally produces the intensional representation for each premise in the problem. An example is shown for ‘The square is on the right of the circle’:

((1 0 0) ‘[]’ ‘O’)

This signifies that the square can be reached from the circle by moving to the right. It is not necessary that the circle is immediately next to the square.

Finally, it has to be mentioned that the all the tokens used in the premises have to be defined in a lexicon; no other tokens may be used in the code. The tokens in the lexicon are: circle, square, triangle, ell, ess, line, cross, vee, star (same tokens as in [3]).

### 3.1.1 Lexicon and Grammar

The parser uses a lexicon and a context-free grammar to produce the intensional representation. The lexicon has the following terminal symbols, along with their syntactic category and semantics:

1. Words which specify directions: such as ‘right’, ‘left’, ‘front’, ‘behind’ and so on. All of these have semantics attached to them: ‘left’ has (0 -1 0) attached to it, for example.
2. Accessory words, such as ‘in’, ‘of’ and ‘to’, which need to be combined with words in category 1. For example, ‘in front of’, ‘to the right of’. The combination is done according to the rules in the context-free grammar.
3. Articles: definite (the) and indefinite (a). However, indefinite articles are not used in the premises.
4. Words corresponding to the tokens: ‘square’, ‘circle’, ‘cross’ and so on. These tokens have suitable semantic representations.

All of the above words are, in addition, assigned a syntactic category.

The context free grammar contains rules to map terminal symbols from the lexicon to non-terminal symbols. In other words, the RHS of the rule is reduced to the LHS. The LHS is always a non-terminal, while the RHS can be a terminal (with syntactic category) or a non-terminal symbol.

### 3.1.2 Spatial Parser code

The Python code implements the Spatial parser through 2 main high-level functions which operate on a stack data structure:

1. Analyze: It implements all the important parts of the parser:
  - (i) Reduces words to their syntactic category and obtains their semantics from the lexicon. Puts them on the stack.
  - (ii) Reduces terminal symbols to non-terminal symbols using the context-free grammar. This involves removing symbols on the RHS from the stack, and inserting the corresponding symbols on the LHS.
2. Shift: which shifts the next token into the stack.

There is also a ‘Backtrack’ high-level function which can be used to ‘reverse’ analyze and shift. However, this is not normally executed while parsing a sentence.

## 3.2 Spatial Reasoning theory: Johnson-Laird/Byrne

The first phase, according to this theory, is the model construction phase, where an initial model is constructed. For verification problems, the conclusion is verified in the initial model. This is followed by a search for counterexamples. If the conclusion is true in the model, a falsification process takes place in which a search is made for an alternative model in which the conclusion is false. On the other hand, if the conclusion is false in the original model, a search is made for an alternative model in which the conclusion is true.

Some of the main functions in the Python code are described at a very high level below. The data structure in which the tokens are inserted is called a *spatial array*.

1. Start model: Called when 2 new entities appear in the premise, neither of which are present in the spatial array.
2. Combine models: Called when 2 separate models have already been created and a new premise appears which relates one token from each of the 2 models together.
3. Add subject or object: A subject (resp. object) is added when a new assertion appears and the object (resp. subject) of this assertion is already in the spatial array, but the subject (resp. object) isn’t. It inserts the subject (resp. object) according to its specified relation with the object (resp. subject).
4. Verify model: Called when a conclusion premise, i.e., a premise in which both the subject and object are already present in the spatial array, appears. If the specified relation between the subject and the object is satisfied, the conclusion is true. Otherwise, it is false.

5. Search for counterexamples: make true and make false: If the function to verify the conclusion returns ‘True’, a function is called to falsify the model. On the other hand, if the function to verify the conclusion returns ‘False’, a function is called to ‘make the model true’. These functions (which produce *one and only one* false/true model respectively if one exists) may return a number of different results for verification problems: false but can also be true, true but can also be false, always true, always false.
6. Generate conclusion (not present in [3]): While the function to verify a conclusion is called for verification problems, a function to generate a conclusion between two tokens is called for generation problems. Once a conclusion is generated, there is a search for counterexamples, like in verification problems. If the same conclusion exists in all possible models, falsification is not possible and therefore only one conclusion (and the initial model) is printed. If there are 2 different conclusions possible, the initial conclusion (and model) and a falsified conclusion (and model) are printed.

### 3.3 PRISM theory and implementation

The PRISM theory builds upon the previous Spatial Reasoning theory in some ways – it has the same functions to start a model, to combine 2 models and to verify a model. However, it differs in the way it tackles indeterminate problems and the way in which it builds alternative models. According to [6], *“This preferred model theory suggests that individuals tackle such indeterminate multiple model problems by constructing a single, simple, and typical mental model, but neglect other possible models. The model that first comes to reasoners minds is the preferred mental model. It helps save cognitive resources, but also leads to reasoning errors and illusory inferences.”*

The theory says that in most cases, only the preferred mental model is constructed. Only in some cases do alternative models have to be constructed. Whenever alternative models need to be searched for, it starts with the preferred model and applies local transformations. Alternative models that require a longer sequence of local transformations are more likely to be neglected than models that are only minor variations of the preferred model.

These local transformations are carried out in the code using the concepts of annotations and neighbourhood graphs. Prism goes through 3 phases: model construction, model inspection and an optional model variation phase.

Annotations are inserted in the model construction phase. Whenever a token can be inserted into more than one position in the spatial array (according to the relation in the premise), it is inserted into the first free position (this strategy is called the *first free fit – fff strategy*) and an annotation is made.

For example, consider Problem 4. Here, as soon as the second premise appears, an annotation is made and C is inserted at the first free position, i.e.

C B A with annotation  $[(-1\ 0\ 0), C, A]$ , i.e. C is *somewhere to the left* of A.

A neighbourhood graph is a graph in which each model is a node and the number of local transformations between two models is the edge distance between two nodes. If applying one local transformation on the preferred model results in an alternative model, the distance is 1. If another local transformation is applied on the model produced by the first transformation, the distance between this model and the preferred model is 2. Prism says that alternative models that are *further away* from the preferred model are more likely to be ignored.

The model construction and inspection (phase which checks if conclusion is true or false/generate a conclusion) phases always occur, while the model variation phase is optional. In this phase, the annotations are processed and alternative models are formed based on them. The order of alternative models is according to their distance in the neighbourhood graph. The Prism Python program implements this by defining a **threshold** (2 by default). Any models at a distance greater than this threshold are never taken into account by the reasoner.

The following set of premises is from Problem 5 in the Prism paper. This is a generation problem with 3



possible models, and the same conclusion exists between the triangle and the start in all of them.

*the square is on the left of the triangle*  
*the circle is on the right of the triangle*  
*the triangle is on the left of the cross*  
*the cross is on the left of the star*  
*what is the relation between the triangle and the star*

In this problem, Prism says that the conclusion in the preferred model is generally accepted. However, it can also look for alternative models by carrying out the model variation phase. The three possible models which are produced are shown in Figure 2. Here, there are two annotations: [(1 0 0) cross triangle], i.e. the cross is to the right of the triangle, and [(100) star cross], i.e. the star is to the right of the cross. The first annotation is processed, and the circle and the cross in the preferred model are swapped to form alternative model 1. To get alternative model 2, the star and the circle in *Alternative model 1* are swapped – this is therefore at a distance of 2 from the preferred model.

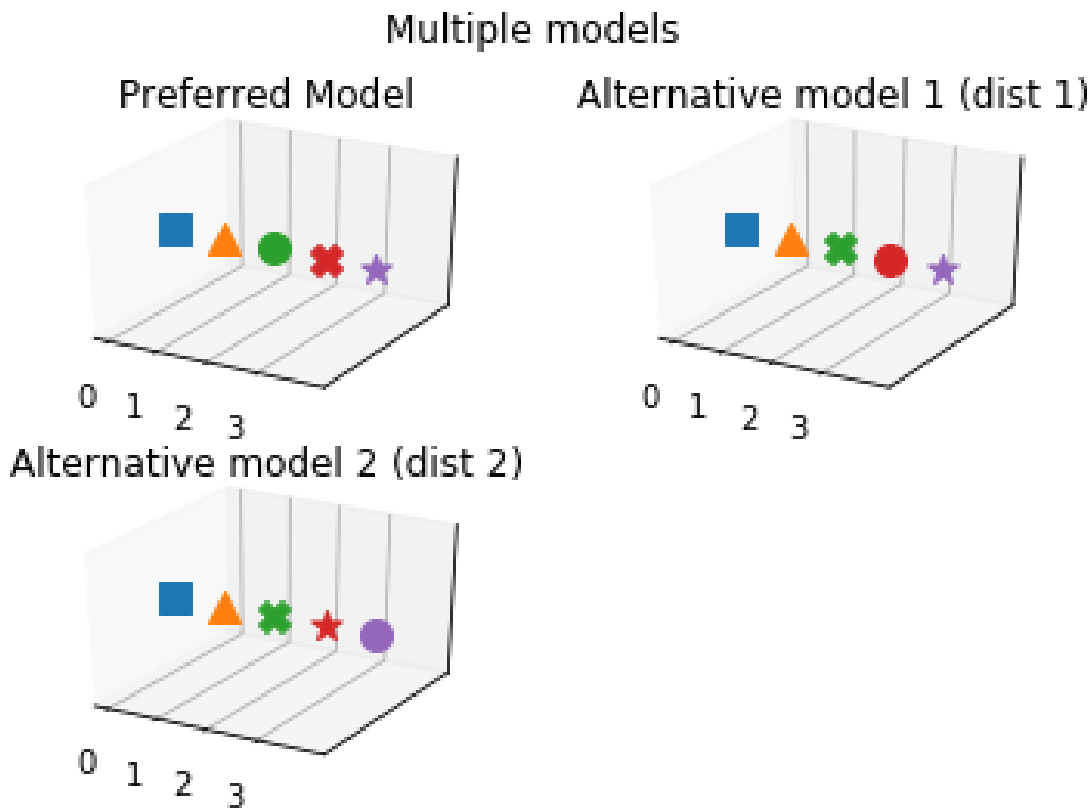


Figure 2: Models according to neighbourhood graph – Prism paper Problem 5

## 4 Scenarios: according to Python implementations of both theories

As mentioned in the previous section, the model variation phase only takes place in certain cases in the Prism python program. Table 1 details the cases in which model variation takes place, and the cases in which it doesn't. The table also gives other important details such as the predicted response and 'validity and correctness of response'. In Table 1, PMM stands for 'Preferred Model' and AMM stands for 'Alternative models'. A similar table for Johnson-Laird/Byrne's classical spatial reasoning theory is shown in Table 2,

where the model variation phase (called ‘search for counterexamples’ phase in this case) always takes place. Note that the ‘generate conclusion’ problems were not present in the Lisp code for this model ([3]).

In both tables, Types 1-3 are determinate, and the rest are indeterminate. ‘Generate all models’ (Type 9) can only be done using Prism, as the framework for Spatial Reasoning (based on the Lisp code [3]) makes it impossible to generate all models. By looking at the tables, we can see that there are some cases where the two theories return different results. It is important to note that Prism explains cases where human reasoners may make mistakes. For example, Type 5 returns an invalid and incorrect result for the Prism code, showing how a human reasoner may make a mistake when a relation holds in the preferred model, but not in the alternative model. Similarly, in type 6, the reasoner may predict ‘Yes’, if (s)he finds a relation that holds in an alternative model within the threshold distance from the preferred model. The classical spatial reasoning theory code, on the other hand, just returns a valid result in both cases.

Table 1: PRISM model: all possible scenarios

Type	Description	Model Variation phase	Predicted Response	Validity and Correctness of response	Comments
1	Verification of a relation that holds in the model, as a valid conclusion	No	Yes	Valid, correct	Prints one and only model
2	Verification of a relation that does not hold in the model, as a valid conclusion	No	No	Invalid, correct	Prints the false model
3	Generation of a relation that is a valid conclusion	No	Conclusion generated	Valid, correct	Prints the generated model along with the conclusion
4	Verification of a relation that holds in all possible models, as a valid conclusion	No	Yes (because it holds in PMM)	Valid, correct	Prints the preferred model: no model variation necessary
5	Verification of a relation that holds in the PMM, as a valid conclusion	No	Yes (because it holds in PMM)	Invalid, incorrect	Prints ONLY the preferred model, no model variation required (alternative models ignored)
6	Verification of a relation that holds in an AMM, but not in the PMM, as a valid conclusion	Yes	Yes (but based on threshold)	Valid, incorrect	Premises are true in an AMM, so true (incorrect). PMM, AMMs printed (upto threshold)

7	Generation of a relation that holds in the PMM and all the AMMs	No (first conclusion is generally accepted without looking at AMMs), but possibly yes.	Conclusion generated	Valid, correct	Same conclusion in all models. Most people accept the first conclusion without looking at AMMs. Models upto threshold printed.
8	Generation of different relations in PMM and AMM.	No (first conclusion is generally accepted without looking at AMMs), but possibly yes.	No Valid conclusion possible	Valid, correct (can be invalid and incorrect if the first conclusion is accepted)	Different conclusions in different models. Most people accept the first conclusion without looking at AMMs. Models upto threshold printed.
9	Generation of all models	Yes (in the order from the neighbourhood graph)	All models generated	Valid, correct	All possible models printed (PMM and AMMs with their distance from the PMM)

Table 2: Johnson-Laird/Byrne’s Spatial Reasoning model: all possible scenarios

Type	Description	Search for counterexamples	Predicted Response	Validity and Correctness of response	Comments
1	Verification of a relation that holds in the model, as a valid conclusion	Yes	Yes	Valid, correct	Prints one and only model
2	Verification of a relation that does not hold in the model, as a valid conclusion	Yes	No	Invalid, correct	Prints the false model
3	Generation of a relation that is a valid conclusion	Yes	Conclusion generated	Valid, correct	Prints the generated model along with the conclusion (not present in [3])
4	Verification of a relation that holds in all possible models, as a valid conclusion	Yes	Yes	Valid, correct	Prints the initial mode : no false model found
5	Verification of a relation that holds in the initial model, as a valid conclusion	Yes	Yes, but can also be no	Valid, correct	When initial model is true, it falsifies it. Prints initial model (true) + 1 falsified model

6	Verification of a relation that holds in an altered model, but not in the initial model, as a valid conclusion	Yes	No, but can also be yes	Valid, correct	When initial model is false, it makes it true. Prints initial model (false) + 1 'made true' model
7	Generation of a relation that holds in all models.	Yes	Conclusion generated	Valid, correct	Only one unfalsifiable conclusion generated: initial model printed.
8	Generation of different relations in different models.	Yes	No Valid conclusion possible	Valid, correct	Initial model and its conclusion printed, along with one falsified/made true model and the corresponding conclusion.
9	Generation of all models	NA	NA	NA	Not feasible using this framework. Only possible using Prism.

## 5 Python Implementation and Execution

This section shows how the code is organized, how it is executed, and describes the types of sample problems mentioned in the code.

### 5.1 Python Module Organization

The Python 3 code is organized into the following modules:

1. `spatial_reasoning.py`: main module which implements Johnson-Laird/Byrne's Spatial Reasoning theory. It calls functions in other modules as required. It also contains the top level functions *test*, *interpret*, *decide* and *call\_appropriate\_func*, along with all the sample problems which will be used to execute both the classic Spatial Reasoning and the Prism programs.
2. `prism.py`: main module which implements the Prism theory. Like `spatial_reasoning.py`, it has its own top level functions *test*, *interpret*, *decide* and *call\_appropriate\_func*, and functions to implement Prism-specific functionality like its method to get alternative models. It also calls functions in the same common modules as `spatial_reasoning` does.
3. `spatial_parser.py`: module containing the major functions of the spatial parser such as *analyze* and *shift*.
4. `parse_helper.py`: module containing auxiliary functions called by the functions in `spatial_parser.py`.
5. `model_builder.py`: module containing functions to start a new model, and to add a subject/object to the model. It also contains Prism's functions to add a subject/object to a model, which include the process of creating annotations.
6. `model_validation.py`: module containing functions for model verification and variation, and other associated functions.
7. `mode_combination.py`: module containing the function for combining two models.
8. `utilities.py`: module containing small utility functions which are called by the other modules (apart from `spatial_parser`, which has its own helper module).
9. `spatial_array.py`: module containing functions which operate directly on the spatial array. Among other things, there are functions to create a new spatial array, expand an existing spatial array, and print spatial arrays in a 3D graph.

## 5.2 Types of sample problems

Sample problems for both theories of Spatial Reasoning are given in `spatial_reasoning.py`. There are different types of problems, and a number of problems of each type. A problem is executed by running either `spatial_reasoning.py` or `prism.py` with the problem type and problem number under that type as command line arguments. The problem types given in the code are:

1. Combination problems: called **‘combination’** (problems no. 1-4) in the code and when called from the command line
2. Inconsistent (conclusion false) verification problems: called **‘inconsistent’** (problems no. 1-3) in the code and when called from the command line
3. Determinate verification problems and indeterminate verification problems with a single conclusion: called **‘deductive’** (problems no. 1-10) in the code and when called from the command line
4. Indeterminate verification problems (different conclusions possible) : called **‘indeterminate’** (problems no. 1-5) in the code and when called from the command line.
5. Determinate generation problems (generate conclusion problems): called **‘generatedet’** (problems no. 1-3) in the code and when called from the command line
5. Indeterminate generation problems (problems where multiple conclusions can be generated): called **‘generateindet’** (problems no. 1-5) in the code and when called from the command line
5. Problems to generate all models: called **‘generateall’** (problems no. 1-12) in the code and when called from the command line. This type of problem should only be run from `prism.py` and not from `spatial_reasoning.py` as Johnson-Laird/Byrne’s Lisp framework cannot be extended to generate all models.

These sample problems include the sample problems in the original Lisp code for the classic spatial reasoning model and the problems defined in the Prism paper [5]. The empirical problems in Experiment 1 of [5] are also included.

## 5.3 Execution

The code is entirely in Python 3, so it has to be executed with a Python 3 interpreter. It will not work with Python 2.x.

### 5.3.1 Classical spatial reasoning model (Johnson-Laird and Byrne)

To execute the Python code for the classical spatial reasoning model (Johnson-Laird and Byrne), one of the following commands is executed based on the operating system:

**Linux/Mac:** `python3 spatial_reasoning.py problem_type problem_no.`

**Windows** (assuming only Python 3.x is installed): `python spatial_reasoning.py problem_type problem_no.`

where

`problem_type`  $\in \{combination, deductive, indeterminate, inconsistent, generatedet, generateindet\}$

`problem_no` is the problem number under the given problem type.

### 5.3.2 Prism Model

To execute the Python code for the Prism model, one of the following commands is executed based on the operating system:

**Linux/Mac:** `python3 prism.py problem_type problem_no.`

**Windows** (assuming only Python 3.x is installed): `python prism.py problem_type problem_no.`

where

`problem_type`  $\in \{combination, deductive, indeterminate, inconsistent, generatedet, generateindet, generateall\}$

`problem_no` is the problem number under the given problem type.

## 5.4 Problems from Tables 1 and 2 in the Python program

The following table (Table 3) gives the problem numbers of example problems of each type mentioned in the Tables 1 and 2. As already mentioned, the ‘generate all’ problems should only be run on prism.py.

Table 3: Examples of Problem types in Tables 1 and 2 in Code

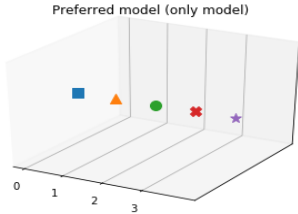
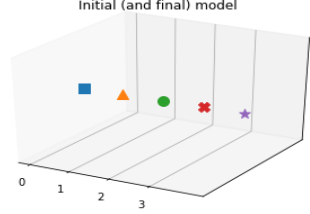
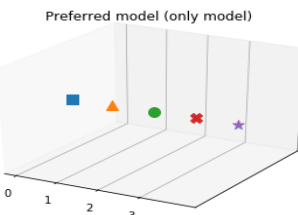
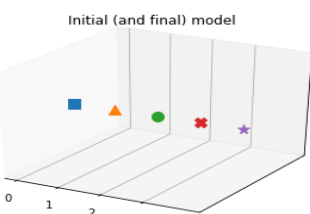
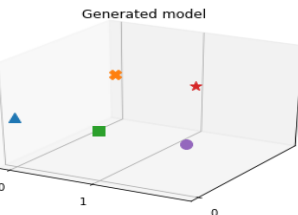
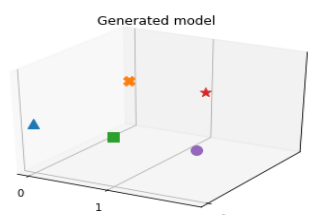
Problem Type from Tables 1 and 2	Example problems: problem type and number in code
1. Determinate verification problem (conclusion true)	deductive 1, deductive 2, deductive 3, deductive 4, deductive 7.
2. Determinate verification problem (conclusion false)	inconsistent 1, inconsistent 2, inconsistent 3
3. Generation of valid conclusion (determinate problem)	generatedet 1, generatedet 2, generatedet 3.
4. Indeterminate verification problem: same conclusion in all models	deductive 5, deductive 6, deductive 8, deductive 9.
5. Indeterminate verification problem: conclusion holds in the initial/preferred model, but not in non-initial/alternative models	indeterminate 1
6. Indeterminate verification problem: conclusion holds in a non-initial/alternative model, but not in the initial/preferred model	indeterminate 2, indeterminate 3, indeterminate 4, indeterminate 5
7. Generation of a relation that in all models	generateindet 3.
8. Generation of different relations in different models	generateindet 1, generateindet 2, generateindet 4, generateindet 5.
9. Generation of all possible models	generateall 1-12

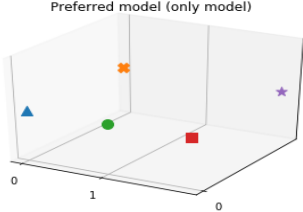
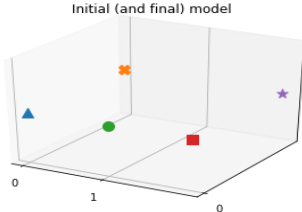
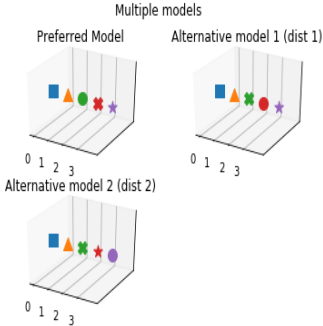
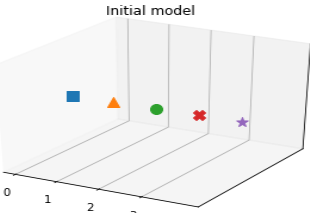
## 5.5 Problems from Prism paper [5] in Python program

Apart from the problems in the Lisp code, all the sample problems in the Prism paper [5] are also executable using the Python programs. These problems are explained in detail in Table 4 – they are executed against the programs for both theories, and detailed results are displayed in this table. Note that the tokens in the Prism paper [5] problems (which are car brands) are changed to the tokens in the lexicon (shapes), but the same relations between the tokens are maintained.

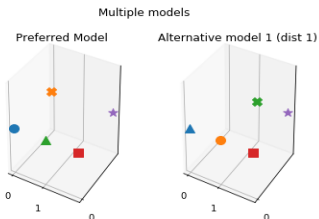
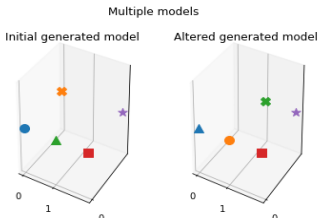
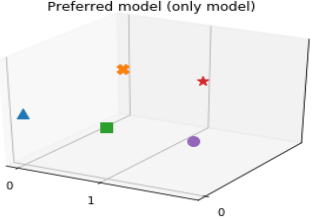
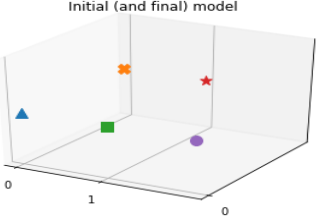
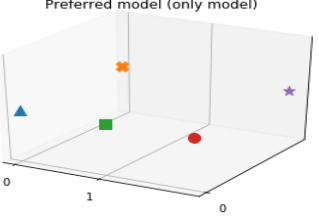
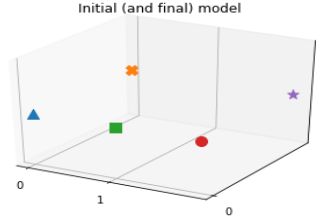
Table 5 contains the empirical problems under Experiment 1 of the Prism paper [5]. All these problems are ‘generate all’ problems and must be executed with prism.py. Again, the tokens (which are letters in [5]) are changed to the tokens present in the lexicon (shapes), but the same relations between the tokens are maintained. Please note that there was a typographical error in the third premise of Experiment 1(b) in the original Prism paper [5], and that has been corrected here (‘behind’ has been corrected to ‘in front of’).

Table 4: Execution of Problems from the Prism paper [5] in both programs

Problem in the Prism paper	Result in Prism code	Result in Classical Spatial Reasoning code	Problem number in Python code
<b>Prism 1</b> <i>(verification, determinate)</i> the square is on the left of the triangle the circle is on the right of the triangle the circle is on the left of the cross the cross is on the left of the star the triangle is on the left of the star	Premises are true in model, NO model variation required. One and only model displayed. 	Premises are true in model. NO falsifying model found. One and only model displayed. 	deductive 7
<b>Prism 2</b> <i>(verification, indeterminate, single solution)</i> the square is on the left of the triangle the circle is on the right of the triangle the triangle is on the left of the cross the cross is on the left of the star the triangle is on the left of the star	Premises are true in model, NO model variation required. 2 models are possible (both with same solution). But as there is no model variation, only 1 is displayed. 	Premises are true in model. NO falsifying model found. Initial model displayed. 	deductive 8
<b>Prism 3</b> <i>(generation, determinate)</i> the circle is on the right of the square the triangle is on the left of the square the cross is in front of the triangle the star is in front of the square what is the relation between the cross and the star	Conclusion generated ( <i>the cross is on the left of the star</i> ), no model variation attempted as there are no annotations to process. Conclusion and model displayed. 	Conclusion generated ( <i>the cross is on the left of the star</i> ), attempt made to falsify it. No model with falsified conclusion found. Conclusion and model displayed. 	generatedet 3

<p><b>Prism 4</b>  <i>(verification, indeterminate, single solution)</i>  the square is on the right of the circle  the triangle is on the left of the square  the cross is in front of the triangle  the star is in front of the square  the star is on the right of the cross</p>	<p>Premises are true in model, no model variation required. 2 models are possible (both with same solution). But as there is no model variation, only 1 is displayed.</p> 	<p>Premises are true in model. NO falsifying model found. Initial model displayed.</p> 	<p>deductive 9</p>
<p><b>Prism 5</b>  <i>(generation, indeterminate, single conclusion)</i>  the square is on the left of the triangle  the circle is on the right of the triangle  the triangle is on the left of the cross  the cross is on the left of the star  what is the relation between the triangle and the star</p>	<p>First conclusion (<i>the triangle is on the left of the star</i>) is generally accepted straightaway. But an annotation is present and Prism can also search for alternative models. 2 AMMs are generated in increasing order of distance from PMM (according to neighbourhood graph, threshold=2). fff is the preferred model. Same conclusion in all the models.</p> 	<p>Model generated, along with conclusion (<i>the triangle is on the left of the star</i>). Attempt to falsify the conclusion by falsifying the obtained model yields no alternate conclusion.</p> 	<p>generateindet 3</p>



<p><b>Prism 6</b>  <i>(generation, indeterminate, no valid conclusion)</i>  the square is on the right of the triangle  the circle is on the left of the square  the cross is in front of the circle  the star is in front of the square  what is the relation between the circle and the triangle</p>	<p>First conclusion (<i>the circle is on the left of the triangle</i>) in generated PMM is generally INCORRECTLY accepted. But an annotation is present and Prism can also search for alternative models. Conclusion in AMM: <i>the circle is on the right of the triangle</i>. Different conclusions in different models: NO VALID CONCLUSION!</p> 	<p>Model generated, along with conclusion (<i>the circle is on the left of the triangle</i>). The conclusion is falsifiable, so 2 models are generated with 2 different conclusions (<i>the circle is on the right of the triangle</i> is also feasible). NO VALID CONCLUSION!</p> 	<p>generateindet 4</p>
<p><b>Prism 7(a)</b>  <i>(verification, determinate)</i>  the circle is on the right of the square  the triangle is on the left of the square  the cross is in front of the triangle  the star is in front of the square  the cross is on the left of the star</p>	<p>Premises are true in model, NO model variation required. One and only model displayed.</p> 	<p>Premises are true in model. NO falsifying model found. One and only model displayed.</p> 	<p>deductive 10</p>
<p><b>Prism 7(b)</b>  <i>(verification, indeterminate, single solution)</i>  the circle is on the right of the square  the triangle is on the left of the circle  the cross is in front of the triangle  the star is in front of the circle  the cross is on the left of the star</p>	<p>Premises are true in model, no model variation required. 2 models are possible (both with same solution). But as there is no model variation, only 1 is displayed.</p> 	<p>Premises are true in model. NO falsifying model found. Initial model displayed.</p> 	<p>deductive 5</p>

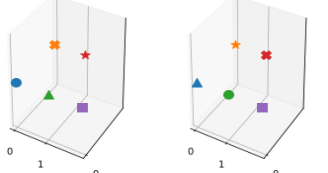
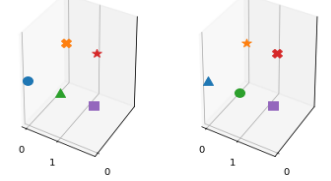
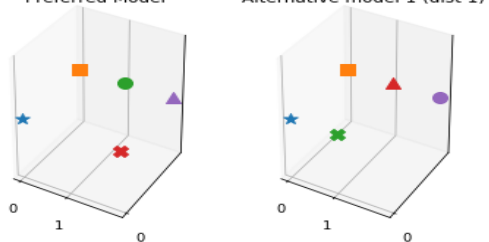
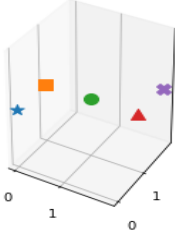
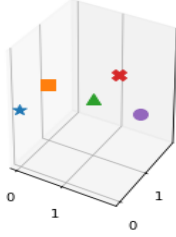
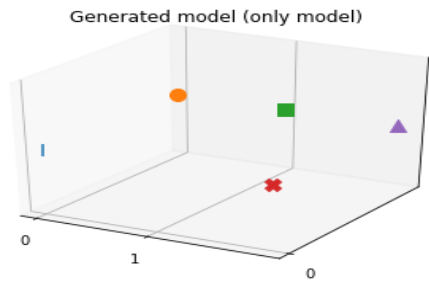
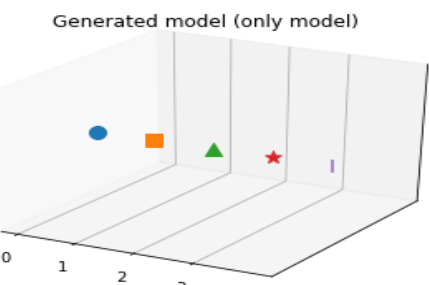
<p><b>Prism 7(c)</b>  <i>(generation, indeterminate, no valid conclusion)</i></p> <p>the square is on the right of the triangle  the circle is on the left of the square  the cross is in front of the circle  the star is in front of the triangle  what is the relation between the cross and the star</p>	<p>First conclusion (<i>the cross is on the left of the star</i>) in generated PMM is generally INCORRECTLY accepted. But an annotation is present and Prism can also search for alternative models. Conclusion in AMM: <i>the cross is on the right of the star</i>. Different conclusions in different models: NO VALID CONCLUSION!</p> <div data-bbox="587 712 916 929"> <p>Multiple models</p> <p>Preferred Model      Alternative model 1 (dist 1)</p>  </div>	<p>Model generated, along with conclusion (<i>the cross is on the left of the star</i>). The conclusion is falsifiable, so 2 models are generated with 2 different conclusions (<i>the cross is on the right of the star</i> is also feasible). NO VALID CONCLUSION!</p> <div data-bbox="946 571 1278 795"> <p>Multiple models</p> <p>Initial generated model      Altered generated model</p>  </div>	<p>generateindet 5</p>
--	--	--	----------------------------

Table 5: Empirical problems from Experiment 1 of the Prism paper [5]

Experiment in the Prism paper	Problem number in Python code	Models generated
<p><b>Experiment 1(a)</b>  the square is on the left of the circle  the triangle is on the right of the square  the cross is behind the triangle  the star is behind the square  generate all models</p>	<p>generateall 8</p>	<p>2 models generated (1 preferred + 1 alternative model)</p> <div data-bbox="962 1317 1453 1601"> <p>Multiple models</p> <p>Preferred Model      Alternative model 1 (dist 1)</p>  </div>

<p><b>Experiment 1(b)</b></p> <p>the square is on the left of the circle  the triangle is on the right of the square  the cross is in front of the triangle  the star is behind the square  generate all models</p>	<p>generateall 9</p>	<p>2 models generated (1 preferred + 1 alternative pre-model)</p> <p>Multiple models</p> <div> <div>Preferred Model</div>  </div> <div> <div>Alternative model 1 (dist 1)</div>  </div>
---	--------------------------	--

<p><b>Experiment 1(e)</b>  the circle is on the left of the square  the triangle is on the right of the square  the cross is behind the triangle  the line is behind the circle  generate all models</p>	generateall 1	1 model generated (preferred model is the only model) 
<p><b>Experiment 1(f)</b>  the square is on the right of the circle  the triangle is on the right of the square  the star is on the right of the triangle  the line is on the right of the star  generate all models</p>	generateall 2	1 model generated (preferred model is the only model) 

## References

- [1] P.N. Johonson-Laird and R Byrne. Spatial reasoning. *Journal of Memory and Language*, 28:564–575, 1989.
- [2] P.N. Johonson-Laird and R Byrne. *Deduction*. Hillside, NJ: Erlbaum, 1st edition edition, 1991.
- [3] Princeton Univeristy. Mental models and reasoning. <http://mentalmodels.princeton.edu/programs/space-6.lisp>. [Online; accessed 13-August-2018].
- [4] Princeton Univeristy. Mental models and reasoning. <http://mentalmodels.princeton.edu/about/what-are-mental-models/>. [Online; accessed 13-August-2018].
- [5] M. Ragni and M. Knauff. A theory and a computational model of spatial reasoning with preferred mental models. *Psychological Review*, 120(3):561–588, 2013.
- [6] Ragni, M., Stahl, P., Seipp, J., Braun, M. Spatial reasoning with prism. <http://imodspace.iig.uni-freiburg.de/prism/>. [Online; accessed 13-August-2018].