Bachelor's Thesis Computing Science

Portia Puzzles

Formalizing and Solving Portia Puzzles Using Logic and SAT Solvers

Noah Romeijnders
s1053637

July 17, 2025

First supervisor/assessor:
dr. Engelbert Hubbers

Second assessor:
dr. Freek Wiedijk

Radboud University

Abstract

Portia puzzles are logic puzzles in which the objective is to identify the correct casket containing a hidden portrait based on a set of statements. In this thesis, we will present a way to formally define Portia Puzzles. We will show two methods of solving Portia puzzles. The first method uses logical deduction. Whereas the second method uses a Boolean Satisfiability (SAT) solver. As a final product, we present a toolkit, that both generates and solves Portia puzzles.

Contents

Chapter 1

Introduction

The class of logic puzzles contains any puzzle for which logical deduction skills are needed to solve them. The most famous logic puzzles include Sudoku, logic grid puzzles and knights and knaves puzzles. Portia puzzles are logic puzzles that are most similar to knights and knaves puzzles. They are both logic puzzles for which the truthfulness of statements is crucial in finding a solution.

This thesis aims to answer the research question. Is it possible to formalize the structure of Portia puzzles in such a way that Portia puzzles can be generated and solved automatically? We do this by splitting the research question up into three sub questions.

- How can you formalize Portia puzzles?
- How can you generate Portia puzzles
- How can you solve Portia puzzles

To solve these sub questions, we introduce a method for encoding these puzzles into boolean formulas, and implement both a logic deduction solver and a SAT-based solver. We also provide algorithm to generate Portia puzzles by hand and automatically using a random puzzle generator.

First, Chapter 2 gives background information about the topics discussed in thesis and aims to answer the question what a Portia puzzle is. Then, Chapter 3 will give two ways to solve Portia puzzles, a logic deduction solver and an SAT solver. In Chapter 4 we discuss how to create Portia puzzles and show some of our own solutions. Chapter 5 will cover other work that is related to our research and give ways to expand our research. Finally in Chapter 6 we will conclude the thesis with the most important takeaways.

Chapter 2

Preliminaries

2.1 Portia caskets

Portia caskets originate from a Shakespeare's *Merchant of Venice*. In this book he introduces three caskets, one gold, silver, and lead. Only one of these caskets contained Portia's portrait. A suitor would need to choose the casket with the portrait to get Portia's hand in marriage. Based on this, Smullyan created the Portia casket logic puzzles, see for instance "What is the name of this book" [16] Instead of the choice based on luck, there are three statements on the casket. These statements tell you which casket contains the portrait, however not all statements are truthful. In the example given by Smullyan's fig. 2.1, exactly one statement is true.

We can deduce the location of the portrait, based on the statements and on the additional knowledge that one statement is true. Let us first assume that the portrait is in the gold casket. This means that statements 1 and 2 are true; namely, it is in the gold casket and not in the silver casket. Two true statements are not possible. Because this contradicts the fact that exactly one statement is true, the assumption must be wrong and the portrait can not be in the gold casket. If we assume that the portrait is in the silver casket, now statements 1 and 2 are false. Statement 3 is true in this case, because the portrait is not in the gold casket but in the silver

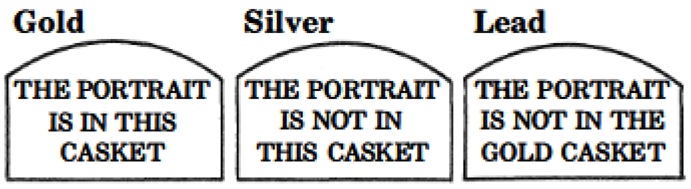
Gold
THE PORTRAIT IS IN THIS CASKET
Silver
THE PORTRAIT IS NOT IN THIS CASKET
Lead
THE PORTRAIT IS NOT IN THE GOLD CASKET

Figure 2.1: Smullyan's Portia Casket

casket. Now we have one true statement. Lastly, if we assume the portrait to be in the lead casket. The first statement is false, but the second and third statements are true. The lead casket also contradicts the fact that exactly one statement is true. Thus, the portrait must be in the silver casket.

Each casket in a Portia puzzle contains some number of n valid statements $n \in \mathbb{Z}^+$. In previous literature like Smullyan[16] and Mackinnon[7] this is limited to one or two. However we allow more statements per casket and will show Portia puzzles with more than 2 statements per casket. The statements that are allowed in this papers' definition of Portia casket puzzle are;

Definition 1. *Valid Statements on casket c for a Portia puzzle*

- *The portrait is in the x casket*
- *The portrait is not in the x casket*
- *The statement(s) on the y casket is(are) true*
- *The statement(s) on the y casket is(are) false*
- *The other statement on this casket is true*
- *The other statement on this casket is false*

We define a valid statement on casket c for a Portia puzzle where $c \in \{gold, silver, lead\}$. We define $x \in \{gold, silver, lead, this\}$. The element $y \in (\{gold, silver, lead\} - \{c\})$. These statements cannot refer to its own casket c. Such a statement on the gold casket can be an element of the set, $(\{gold, silver, lead\} - \{gold\}) = \{silver, lead\}$. The last two statements are prohibited on caskets with one statement. It is also good to note just because a statement such as *The statement on the gold casket is true* does not necessarily mean that the statement holds. This statement can be false, and thus the statements on the gold casket must be false.

2.2 Formal Definition of Portia puzzle

In "What is the name of this book" [16], Smullyan does not give a formal definition of Portia puzzles. This is why we will give a definition that we will use for all Portia puzzles in this paper.

A valid Portia casket puzzle is defined as follows:

Definition 2. *Portia-n-m*

- *where $n \in \mathbb{Z}^+$, and where $0 \leq m \leq 3 \cdot n$*

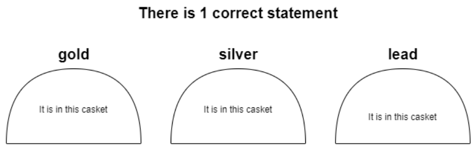

Figure 2.2: Portia-1-1 puzzle

- *Each casket must contain exactly n statements*
- *It must have exactly three caskets, the first gold, the second silver, the third lead*
- *There are m true statements*
- *All statements on the caskets must be valid statements as defined by Definition 1*
- *There is exactly one portrait, and this portrait is in one of the three caskets.*

To be considered valid, a Portia puzzle's statements must only allow for one casket to contain the portrait. A Portia puzzle that allows for multiple or no solutions are not valid.

Using this Definition 2, we can now create valid and invalid Portia puzzles. You can now categorize a Portia puzzle like fig. 2.2 . This would be a Portia-1-1 puzzle. You can now also check whether fig. 2.2 is a valid puzzle.

The fig. 2.2 has exactly three caskets, each of these caskets contain exactly one statement and all statements are in Definition 1. Now we check whether this Portia puzzle allows for multiple solutions. We assume the portrait is in every casket. If the portrait is in the gold casket, there is one true statement, namely its own statement. This is the same for the silver and lead caskets. Therefore, fig. 2.2 is not a valid Portia-1-1 puzzle because it allows for all caskets to be a valid solution.

We can now also show that fig. 2.1 is a valid Portia-1-1 puzzle. This puzzle has exactly three caskets, each of these caskets contain exactly one statement and all statements are in Definition 1. We have already shown that fig. 2.1 only has one solution, thus this fig. 2.1 is a valid Portia-1-1 puzzle.

The Portia puzzle fig. 2.3 is another unique case. This Portia-1-0 puzzle fig. 2.3 is not a valid Portia puzzle because there is no solution. There is no casket for which there are zero true statements. If the portrait were in the gold casket, there would be two true statements. If we assume that

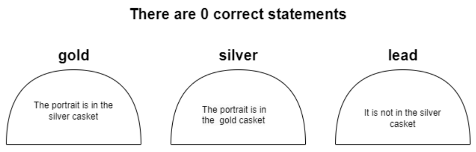
There are 0 correct statements
gold
silver
lead
The portrait is in the silver casket
The portrait is in the gold casket
It is not in the silver casket

Figure 2.3: Portia-1-0

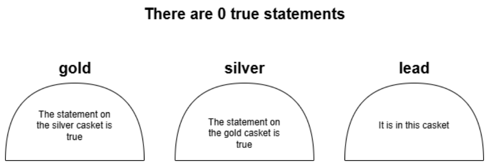
There are 0 true statements
gold
silver
lead
The statement on the silver casket is true
The statement on the gold casket is true
It is in this casket

Figure 2.4: Portia-1-0

the portrait is in the silver casket, there is one true statement; similarly, if we assume that the portrait is in the lead casket, there is also one true statement. To make fig. 2.3 a valid Portia puzzle, one must change the amount of true statements to two. The new Portia-1-2 would be valid with the portrait being in the gold casket.

The Portia puzzle fig. 2.4 displays a non-valid Portia puzzle. The statements, *The statement on the silver casket is true* , on the gold casket and, *The statement on the gold casket is true*, on the silver casket. Calculating the truthfulness of the gold statement requires the statement on the silver casket to be true. The silver casket's statement is only true if the gold statement is true. This repeats infinitely, thus you are unable to assess if these statements are true. We can conclude that statements about the truthfulness can not recursively reference each other.

2.3 Different Portia Puzzles

Definition 2 of Portia Puzzles allows us to calculate and construct every different Portia Puzzle. You might expect the number of Portia-1-m puzzles is $4 * 3 * 8^3 = 6144$. The number of statement combinations multiplied by the number of positions the portrait can be and by how many statements are true. However, many of these Portia-1-m puzzles are invalid due to the constraints that there must be a unique solution. The only way to construct every valid Portia-n-m puzzle is by doing a brute force check if everyone of the 6144 puzzles is valid. In a blog post by Dan Mackinnon [7], this has already done this. The author loops through the list of all statement possibilities and check how many statements are true assuming the portrait is in a specific casket. Each puzzle that gives a truth count that is unique for that statement combinations corresponds to a valid puzzle. They generate

348 valid Portia Puzzles [7] with one statement per casket and, 19752 for Portia puzzles with two statements.

2.4 Natural Language Parsing

Natural Language Parsing (NLP) is a field all about teaching machines how to understand human languages and extract meaning from text. NLP has been used to solve logic grid puzzles [9]. That paper uses NLP concepts extracting the meaning of comparative structures [2]. As it seems like they have a similar objective, it makes sense to investigate their method.

The LGPSolver from [9] uses DistilBERT [15]. DistilBERT is a way to train a neural network that builds on BERT. These neural networks need a lot of training data to achieve good performance. The valid statements as defined in Definition 1 only allow for sixteen different statements. This is definitely not enough training data to create a usable neural network. Our implementation would definitely suffer from overfitting [5]. Overfitting is when a model memorizes the training data instead of learning general patterns. This can happen due to the training dataset being too small. With a small dataset it might also be case that when you split the data into train and test some of the feature are only in the training data or only in the testing.

The method used in [9] is not appropriate in our case. A Google search suggested we use a regular expression (regex) operations and it seems that using regexes is great for recognizing patterns and extracting keywords from text [8]. Regex does not need training data and does not learn like a DistilBERT based model.We found that regex works well for our Statements. We have therefore decide to use regex.

2.5 SAT

The Boolean Satisfiability problem asks whether there exist a valuation of the boolean formula such that it evaluates to true. The Boolean Satisfiability problem is often abbreviated to SATISFIABILITY or SAT. SAT is a known NP-complete problem meaning there is no known algorithm that efficiently solves each SAT problem. Nevertheless SAT is one of the most extensively researched problems in computer science[4]. SAT solvers are being improved constantly as can be seen in the annual SAT Competition[17].

Chapter 3

Research

3.1 Plain Text Representation of Portia Puzzles

There is no default way to specify a Portia Puzzle in only natural language. The way Smullyan does it in "What is the name of this book" [16] is making a graphic for every different puzzle, such as fig. 2.1. In this thesis we formally define a way to present all Portia puzzles using only natural language. There are two parts to natural language representation of Portia puzzles. The first is to define what Portia-*n*-*m* we are dealing with. The second part is all the statements. Each statement is written on a new line. These statements are ordered with the first n statements belonging to the gold casket, the next n statements are on the silver casket, and the last n statements are on the lead casket. This is the natural language representation of fig. 2.1.

```
Portia 1, There are 2 true statements
The portrait is in this casket
The portrait is not in this casket
The portrait is not in the gold casket
```

The first line specifies that this is a Portia-1-1 puzzle. The gold casket has the statement, *The portrait is in this casket.* The silver casket has the statement, *The portrait is not in this casket.* The lead casket has the statement, *The portrait is not in the gold casket.*

3.2 Plain Text Parser

This natural language representation makes it easy to write and formulate Portia puzzles. However, it has a lot of unnecessary information. This where a parser comes in. You take the plain text and transform this in to only the essential information that is needed to solve the puzzle. We used regular expression operations[8] to transform the plain text into JSON that is then used to find a solution.

The parser uses the six sentences as defined in Definition 1 and creates the three patterns to distinguish between sentences and collect the necessary information. These patterns match the words that are the same between sentences, and take out the words that are different and important. Statement_pattern takes sentences like, *The portrait is not in the gold casket* and *The portrait is in this casket.* The information that we need to solve the puzzle is which casket the statement is about and if the portrait is or is not in that casket. The expression truth_pattern takes sentences like, *The statements on the lead casket are true* and *The statements on the silver casket are false.* We extract which casket the statement is about and if that casket is true or false. The expression other_pattern takes sentences like, *The other statement on this casket is false* and *The other statements on this casket are true.* We want to know what casket it is about and if its statements are true or false.

```
    statement_pattern = re.compile(r"The portrait is (not )?in (?:the (\w+)|this) casket")
    truth_pattern = re.compile(r"The statement(?:s)? on the (\w+) casket (?:is|are) (true|false)")
    other_pattern = re.compile(r"The other statement(?:s)? on this casket (?:is|are) (true|false)")
```

These patterns allow you to create a JSON file with the important information. This is the parsed data for fig. 2.1.

```
{
    "portia": 1,
    "true_statements": 1,
    "caskets": {
        "gold": [
            {
                "contains_portrait": true,
                "mentioned_casket": "gold"
            }
        ],
        "silver": [
            {
                "contains_portrait": false,
                "mentioned_casket": "silver"
            }
        ],
        "lead": [
            {
                "contains_portrait": false,
                "mentioned_casket": "gold"
            }
        ]
    }
}
```

We can see that portia : 1 and true_ statements : 1 means that it is a Portia-1-1 puzzle. We can see that the gold casket has a statement that claims that the gold contains a portrait.

3.3 Portia puzzle solver

One of the ways we solved Portia puzzles was to use a SAT solver. This SAT solver takes CNF formulas, therefore we first had to convert our JSON representation into CNF formulas.

3.3.1 Convert Portia puzzle to CNF

First we give definitions of the literals we will use.

Definition 3. *Literal representation: meaning*

- G *: portrait is in gold*
- S *: portrait is in silver*
- L *: portrait is in lead*
- n_i *: statement n*

Where $i \in \mathrm{N}^+$

Literals are variables that can be true or false. Thus in a valid solution, if G is True, that means the portrait is in gold, and S and L must be False for a valid solution. A statement is represented with an integer where n_i corresponds to the truth value of statement n_i.

The first part are the requirements for a valid puzzle. *There is exactly one Casket that contains the Portrait* translates to:

$$(G \vee S \vee L) \wedge (\neg G \vee \neg S) \wedge (\neg G \vee \neg L) \wedge (\neg S \vee \neg L)$$

$(G \vee S \vee L)$ ensures that at least one portrait literal must be true. Whereas $(\neg G \vee \neg S) \wedge (\neg G \vee \neg L) \wedge (\neg S \vee \neg L)$ ensures that no two portrait literals can be true. Together these ensure that the requirement *There is exactly one Casket that contains the Portrait* is held. The only way to get a true value is for one and only one portrait literal to be true.

The second requirement is *Portia-n-m There is exactly m true statements.* We first need to account for two edge cases namely $m = n$ and $m = 0$. These correspond to all statements are true or all statements are false. These two cases are trivial and are converted to CNF by making a conjunction of all statements. *Portia-n-m There is exactly* n *true statements* is converted to:

$$(n_1) \wedge (n_2) \wedge \cdots \wedge (n_m)$$

The statement *Portia-n-m There is exactly* 0 *true statements* is converted to:

$$(\neg n_1) \wedge (\neg n_2) \wedge \cdots \wedge (\neg n_m)$$

To convert the other cases into CNF we split them into two parts at least m are true and at most m are true.

at most m statements are true No more than m statements are allowed to be true. In other words in a subset of size $m+1$ statements, at least one statement must be false. If we take all subsets of size $m+1$ and we combine the parts as a disjunction of the negated elements of such a subset together $(\neg n_1 \vee \neg n_2 \vee \cdots \vee \neg n_m \vee \neg n_{m+1})$. Then we conjunct all of the subsets together.
In the example where $n = 3$ and $m = 2$,this would result in the formula.

$$(\neg n_1 \vee \neg n_2 \vee \neg n_3)$$

This only returns false if all statements are true thus the premise, at most m statements are true, holds.

at least m statement are true In other words, no more than $n-m$ statements are allowed to be false. Therefore we cannot allow $n-m+1$ statements to all be false. We enforce that for every subset of size $n-m+1$ at least one statement must be true. We disjunct the elements of each subset of size $n-m+1$ together. $(n_1 \vee n_2 \vee \cdots \vee n_{n-m} \vee n_{n-m+1})$ Then we conjunct all of the subsets together.
In the example where $n = 3, m = 2$,this would result in the formula.

$$(n_1 \vee n_2) \wedge (n_1 \vee n_3) \wedge (n_2 \vee n_3)$$

When we add these two together, this gives exactly m statements are true because at least m are true and at most m are true.

3.3.2 Encode statements

The final part is the encoding of the statements in CNF formulas. The statement *The portrait is in casket c* would be encoded as.

$$(\neg n_i \vee c) \wedge (n_i \vee \neg c)$$

This ensures that the statement is true if and only if portrait is in c. The only valid combinations are if both literals are true or both false.

The statement *The portrait is not in casket c* is encoded as.

$$(\neg n_i \vee \neg c) \wedge (n_i \vee c)$$

This ensures that only when $n_i \neq c$ the formula can be true. The $(\neg n_i \vee \neg c)$ does not allow both to be true and $(i \vee c)$ does not allow both to be false.

The statement *The statements on casket c are true* is encoded as. This statement can only be if and only if all statements on c are true.

$$\begin{aligned}&n_i \leftrightarrow (n_{c1} \wedge n_{c2} \wedge \cdots \wedge n_{cn}) \equiv\\&(n_i \rightarrow (n_{c1} \wedge n_{c2} \wedge \cdots \wedge n_{cn})) \wedge ((n_{c1} \wedge n_{c2} \wedge \cdots \wedge n_{c3}) \rightarrow n_i)\\&(\neg n_i \vee n_{c1}) \wedge (\neg n_i \vee n_{c2}) \wedge \cdots \wedge (\neg n_i \vee n_{cn}) \wedge (n_i \vee \neg n_{c1} \vee \neg n_{c2} \vee \cdots \vee \neg n_{cn})\end{aligned}$$

Where n_i refers to the statement *The statements on casket c are true.* The terms n_{ci} refer to the ith statement on the casket c.

When you convert $(n_i \rightarrow c)$ to CNF you get $(\neg n_i \vee n_{c1})$, and $((n_{c1} \wedge n_{c2} \wedge \cdots \wedge n_{cn}) \rightarrow n_i)$ becomes $(n_i \vee \neg n_{c1} \vee \neg n_{c2} \vee \cdots \vee \neg n_{cn})$. When we convert this entire statement to CNF form we get:

$$(\neg n_i \vee n_{c1}) \wedge (\neg n_i \vee n_{c2}) \wedge \cdots \wedge (\neg n_i \vee n_{cn}) \wedge (n_i \vee \neg n_{c1} \vee \neg n_{c2} \vee \cdots \vee \neg n_{cn})$$

For example, let us consider a Portia-2-1 puzzle with statement n_1 *The statements on the silver casket are true.* We know that each casket has two statements due to Portia-2-1. The statements on the silver casket are $\{n_3, n_4\}$. The CNF formula of this would be:

$$(\neg n_1 \vee n_3) \wedge (\neg n_1 \vee n_4) \wedge (n_1 \vee \neg n_3 \vee \neg n_4)$$

The statement *The statements on casket c are false* is encoded as.

$$\begin{aligned}&n_i \leftrightarrow \neg(n_{c1} \wedge n_{c2} \wedge \cdots \wedge n_{cn}) \equiv\\&(n_i \rightarrow (\neg n_{c1} \wedge \neg n_{c2} \wedge \cdots \wedge \neg n_{cn})) \wedge ((\neg n_{c1} \wedge \neg n_{c2} \wedge \cdots \wedge \neg n_{cn}) \rightarrow n_i) \equiv\\&(\neg n_i \vee \neg n_{c1}) \wedge (\neg n_i \vee \neg n_{c2}) \wedge \cdots \wedge (\neg n_i \vee \neg n_{cn}) \wedge (n_i \vee n_{c1} \vee n_{c2} \vee \cdots \vee n_{cn})\end{aligned} \tag{3.1}$$

Where n_i refers to the statement *The statements on casket c are true.* The terms n_{ci} refer to the ith statement on the casket c.

When you convert $(n_i \rightarrow \neg n_{ci})$ to CNF you get $(\neg n_i \vee \neg n_{c1})$, and $(\neg(n_{c1} \wedge n_{c2} \wedge \cdots \wedge n_{cn}) \rightarrow n_i)$ becomes $(n_i \vee n_{ci} \vee n_{c2} \vee \cdots \vee n_{cn})$. This is similar to the *The statements on casket c are true* CNF formula except, an extra negation is added before each casket statement n_{cn}. We also simplify $\neg\neg n_{cn} \equiv n_{cn}$. When we convert this entire statement to CNF form we get:

$$(\neg n_i \vee \neg n_{c1}) \wedge (\neg n_i \vee \neg n_{c2}) \wedge \cdots \wedge (\neg n_i \vee \neg n_{cn}) \wedge (n_i \vee n_{c1} \vee n_{c2} \vee \cdots \vee n_{cn})$$

The statements *The statements on casket c are true* have almost the same formulas as *The other statements on this casket are true.* The formula for *The statements on casket c are true* is $n_i \leftrightarrow (n_{c1} \wedge n_{c2} \wedge \cdots \wedge n_{cn})$. This formula changes to $n_{c1} \leftrightarrow (n_{c2} \wedge \cdots \wedge n_{cn})$ where n_{c1} is the statement *The other statements on this casket are true.* The statement *The statements on the casket silver casket are true* on a Portia-2-1 has the formula.

$$(\neg n_1 \vee n_3) \wedge (\neg n_1 \vee n_4) \wedge (n_1 \vee \neg n_3 \vee \neg n_4)$$

If we have the statement *The other statements on this casket are true* as the first statement on the silver casket of a Portia-2-1. We get the boolean formula.

$$(\neg n_3 \vee n_4) \wedge (n_3 \vee \neg n_4)$$

These two statements *The statements on casket c are true* and *The other statements on this casket are true* are follow from a similar base formula and only really differ in the statement count and numbering.

When we combine all the above discussed parts we have a Portia-n-m defined as CNF formula. Using this we created a CNF formula from Smullyan's First Portia Puzzle Figure 2.1.

Listing 3.1: CNF representation of Smullyan's First Portia Puzzle

```
(G \/ S \/ L) /\ # There is exactly one Casket that contains the
      Portrait
(~G \/~S) /\
(~G \/ ~L) /\
(~S \/ ~L) /\
(n1 \/ n2 \/ n3) /\ # at least 1 statement is true
(~n1 \/ ~n2) /\ # at most 1 statement is true
(~n1 \/ ~n3) /\
(~n2 \/ ~n3) /\
(G \/ ~n1) /\  # statement 1
(~G \/ n1) /\
(S \/ n2) /\ # statement 2
(~S \/ ~n2) /\
(G \/ n3) /\ # statement 3
(~G \/ ~n3)
```

3.3.3 CNF Parser

We constructed a Python program to automate the parsing of Portia-n-m to CNF Formulas. The program generates the necessary CNF clauses based on the previous definitions. The program translates the Portia-n-m from JSON format to CNF formula. The following section provides a detailed breakdown of how the code works, and the entire code is available in [13].

The requirement *Portia-n-m There is exactly m true statements* gets converted using the code below. The first two cases namely, $m = 0$ and $n =$

m correspond to all statements are false and all statements are true. These are special cases because the method of taking at least m and at most m does not work. These cases are trivial and are either all statements negated or all statements respectively. The function combinations(statement_ids, n) creates a list with all subsets from statement_ids of size n.

```
    if m == 0:
        for all_false in combinations(statement_ids, n):
            n_m_true_statements.append([f"~{v}" for v in
                all_false])

        return create_cnf_string(n_m_true_statements)

    if n == m:
        for all_false in combinations(statement_ids, n):
            n_m_true_statements.append([f"{v}" for v in
                all_false])

        return create_cnf_string(n_m_true_statements)


    # atleast n are true
    for atleast_vars in combinations(statement_ids, n - m + 1):
        n_m_true_statements.append([f"{v}" for v in atleast_vars
            ])
    # atmost n are true
    for atmost_vars in combinations(statement_ids, m + 1):
        n_m_true_statements.append([f"~{v}" for v in atmost_vars
            ])

    return create_cnf_string(n_m_true_statements)
```

The statement *The statements on casket c are true* and *The statements on casket c are false* is handeled using this function. The variable statement_id_mentioned_casket is a list of the ids corresponding to the referenced casket c in the statement.

```
def truth_pattern_statement_to_cnf (statement, statement_id,
    portia_n):
    mentioned_casket = statement["mentioned_casket"]
    statement_id_mentioned_casket = statement_id_caskets(
        mentioned_casket, portia_n)
    if(statement["statement_truth" ] == "true"):
        x = create_cnf_string([[ n , "~" + statement_id] for n
            in statement_id_mentioned_casket])
        temp = statement_id_mentioned_casket
        temp = ["~" + n for n in temp]
        temp.append(statement_id)
        return " /\\\n" + x + " /\\\n" + cnf_string(temp)

    else :
        x = create_cnf_string([[  "~" + n , "~" + statement_id]
            for n in statement_id_mentioned_casket])
        temp = statement_id_mentioned_casket
        temp.append(statement_id)
        return " /\\\n" + x + " /\\\n" + cnf_string(temp)
```

This line outputs $(\neg n_i \vee c_1) \wedge (\neg n_i \vee c_2) \wedge \cdots \wedge (\neg n_i \vee c_n)$.

```
        x = create_cnf_string([[ n , "~" + statement_id] for n
            in statement_id_mentioned_casket])
```

The following three lines

```
        temp = statement_id_mentioned_casket
        temp = ["~" + n for n in temp]
        temp.append(statement_id)
```

correspond to $(n_i \vee \neg c_1 \vee \neg c_2 \vee \cdots \vee \neg c_n)$. Adding both these parts together converts the statements *The statements on casket c are true* to CNF.

3.3.4 SAT Solver

With the CNF representation now established we can use a SAT solver to solve Portia Puzzles. A SAT solver aims to find a solution that satisfies the formula, find a valuation for the variables such that the whole formula is true. In our case if the whole formula is true we have found a solution to the Portia puzzle. A valid valuation for the formula at listing 3.1 would be $\{G: 0, S: 1, L: 0, n1: 0, n2: 0, n3: 1\}$. You can check that this valuation returns sat,that it is satisfiable, because all disjunction clauses return true. The valuation $\{G: 0, S: 1, L: 0, n1: 0, n2: 0, n3: 1\}$ would correspond to the solution the portrait is in the silver casket and only the third statement is true.

The SAT solver we are using is the Glucose, version 3 [1] via the PySAT module. Glucose is a well known SAT solver that has been top scoring in SAT competitions [17]. Glucose has been updated to newer versions, we still use the version 3 because we are using model viewing to output the valuations which works better with the Glucose version 3.

3.3.5 Logical Deduction Solver

We also constructed a logical deduction solver. The logical deduction solver uses the way we have been using to solve the Portia puzzles by hand as in section 2.1. It assumes a casket to be true and then calculates the number of true statements that follow from that assumption. If the number of true statements is equal to the number specified by the puzzle you have found the solution.

The Logical deduction Solver takes as input the JSON representation of Portia Puzzles discussed in section 3.2. The actual full code can be found on GitHub [13]. In particular in the full code you can see how the function Statement_True(c,s) is handled for every type of statement.

Algorithm 1 Logical deduction Solver

```
for c in caskets do
    $count = 0$
    for s in statements do
        if Statement_True(c,s) then
            $count++$
        end if
    end for
    if count == number_true_statements then
        Return c
    end if
end for
```

Chapter 4

Results

To evaluate the consistency of our formalization and solvers, we give 25 Portia puzzles. These can be found on GitHub [13]. this set of puzzles puzzles consist of 10 Portia-1-m, 8 Portia-2-m puzzles, 5 Portia-3-m puzzles and 2 Portia-4-m puzzles. We have given multiple puzzles with differences in the number of true statements m and an number of statements per casket. We also give novel Portia puzzles with three and four statements, for which we could not find any other examples in literature. The first Portia-1-m and Portia-2-m are taken from Smullyan[16]. The rest of the test cases are new Portia puzzles created by hand by us as explained in Section section 4.1. When running these test cases both the Logic Deduction Solver and SAT solver output the same solutions. We also provide a toolkit for generating and solving Portia-n-m. This program allows a user to to input the natural language representation of a Portia puzzle. If this representation complies with the rules given in Definition 1 and Definition 2, it will output both the Logic Deduction Solution and the SAT solution.

4.1 Creating Portia-n-m

We will now examine the methodologies involved in the construction of new Portia-n-m puzzles. To do this, we created a non-deterministic algorithm to simplify the creation of Portia-n-m puzzles.

Algorithm 2 Create Portia-n-m

```
$s \leftarrow 3 \cdot n, \{n \in \mathbb{Z}^+\}$
$m \leftarrow \{0 \leq m \leq s\}$
$c \leftarrow$ portrait_location()
true_count $\leftarrow 0$
add_other_statements()
while number(statements) < s do
    add_statement()
    update(true_count)
    if true_count $> m$ then
        remove statements or increase m
    end if
end while
no_multiple_solutions(statements)
```

This algorithm can be used to create of Portia-*n*-*m* puzzles. First we decide what type of puzzle we want to create by defining n, m, and the casket that contains the portrait. Defining these variables makes it possible to calculate the number of true statements and to check if they match with your goal. The first statements we want to add are the statements about the other statements like *The statements on the gold casket are true.* We want to add these statement first because these statements depend on other statements if they are true. You have to pay extra attention to statements you add to the corresponding casket. Then you add all of the portrait statements. After adding a statement you calculate the number of true statements. If the number of true statements is greater than the chosen m. You either increase m or remove the statement. Once no further statements can be added to the new Portia-*n*-*m* puzzle, you still need to make sure you only have one solution. The function no_multiple_solutions (statements) uses the logical deduction solver method described in algorithm 1. The function no_multiple_solutions (statements) checks if there are no more locations of the portrait that result in m true statements.

We now demonstrate a run of the algorithm by constructing a new Portia-*n*-*m* puzzle. First we choose to make a Portia-2-4 with the portrait in the silver casket. After deciding the variables we add two statements *The statements on the gold casket are true* on the silver casket and *The other statement on the this casket is true* on the lead casket. After adding these statements we add *The portrait is not in the lead casket* and *The portrait is in the silver casket.* We now need to calculate the the number of true statements. We find that the number of true statements is three, the statements on the gold casket and the silver casket. With this knowledge, we know that only one other statement can be true to achieve the four true statements defined by m. We also know that this true statement cannot be on the lead casket because of the already defined statement *The other statement on the this casket is true.* Therefore we add the statement *The portrait is not in the gold casket* to the silver casket and add *The portrait is in the gold casket* on the lead casket. Now that we have added all the statements we need to check that the silver casket is the only solution. This is the case because when you assume the gold casket you have three true statements and you have 1 true statement if you assume the lead casket. Now we have created a new Portia-2-4.

Example 1. *A new Portia*-2-4

```
Portia 2, There are 4 true statements
The portrait is not in the lead casket
The portrait is in the silver casket
The portrait is not in the gold casket
The other statement on this casket is true
The portrait is in the gold casket
```

While the example was primarily introduced to demonstrate the creation algorithm, it also reveals an interesting property. The exact same statements can be used to create three distinct Portia puzzles.

- A Portia-2-3 with the gold casket as the solution
- A Portia-2-1 with the lead casket as the solution
- A Portia-2-4 with the silver casket as the solution.

The property that the same statements can create multiple distinct valid Portia puzzles can be used to quickly generate multiple Portia puzzles. Using the same method as in constructing Portia puzzles [7], you can generate Portia puzzles by trying random valid statement combinations and using the logical deduction solver to see if it has solutions. We prefer to use the algorithm discussed here because it allows you to create custom solutions and add interesting statement combinations.

4.2 Generating Portia-n

We have shown an algorithm to create Portia-*n*-*m* puzzles. This algorithm 2 is great for creating a new Portia-*n*-*m* puzzle by hand. However it is also possible to randomly create a new Portia-*n* puzzle. We generate a new Portia-*n* puzzle using the algorithm. This algorithm only takes the number of statements per casket *n*. It is not defined how many true statements *m* there are at runtime. This number of true statements *m* can change in different runs of the algorithm.

Algorithm 3 Generate Portia-n (n, valid_statements)

```
if n = 1 then
    Remove_other_statements( valid_statements)
end if
valid_statements[casket] = Remove_truth_statements(valid_statements)
for casket in [gold,silver,lead] do
    statement_list += Random_choice(valid_statements[casket], n)
    Remove_looping_statements(statement_list, Valid_statements)
end for
m = solve(Portia-n)
Portia-$n$-$m$ = create_puzzle(n, m, statement_list)
return Portia-$n$-$m$
```

This algorithm takes the number of statements per casket and the valid statements described in Definition 1. First we remove the statements *The other statements on this casket are true/false* if there is only one statement per casket. These statements are not allowed if you only have one statement per casket. Then we filter out all the statements not allowed on a specific casket to create a list of valid statement per casket. This gets stored in valid_statements[casket]. With this we can loop over all the caskets and randomly choose n statements from valid_statements[casket]. After we have chosen the statements that will go on a specific casket we must remove all statements that can cause a loop. As these loops result in a puzzle not being able to be solved. We check for these loops by creating a tree where the parent is the current node and the child is the referenced node. We add this new reference node to all the nodes that equal the parent casket. We can then run depth first search on this tree and if we encounter a node twice in a path we have a loop. When we have the statements we want per casket we can still need to decide the m. This is done using a variation of the solving algorithm 1. We assume that the portrait is in a casket and count the number of true statements that follow from this assumption. We do this for all caskets and choose a unique m. When we combine the given n, the randomly created statements and the calculate unique m we have created a new Portia-n-m puzzle.

Using this algorithm we were able to create different Portia-8-m, Portia-9-m and Portia-10-m. Using the valid statements provided in Definition 1 it is currently impossible to create a Portia-11-m that does not use a statement more than once and is solvable. Although this Portia-10-m is quite trivial because it use almost every statement. This algorithm is good and creating puzzles with five, six, seven or eight statements. These puzzles are challenging to create using the manual algorithm 2 but the generator is able to generate them comfortably.

Chapter 5

Related and Future Work

5.1 Formalizing Logic Puzzles

The formalization of puzzles have been explored using a variety of methods. Bertier formalizes Sudoku and some other puzzles as a constraint satisfaction problem [3]. Itsuki Maeda and Yasuhiro Inoue give a mathematical foundation for many basic concepts used in logic puzzles [12]. Hans van Ditmarsch and Ji Ruan transformed What Sum logic puzzles into modal logic [6].

5.2 Using SAT solvers on logic puzzles

Transforming logic puzzles into a boolean satisfiable problems has been done. Various puzzle have been converted to a SAT problem, for example the work of Lynce and Ouaknine [11] transforms the Sudoku puzzle into CNF. SAT solvers have also been used to solve path puzzles [14]. Other bachelor students at the Radboud University have also transformed puzzles into SAT. Nick van Oers transformed Pattern into a SAT problem [18]. Laura Kolijn turned Skyscraper Puzzles into SAT [10].

5.3 Enhanced Portia Puzzles

When someone enhances, adds or bends a logical puzzle one can get a better conceptual knowledge of the problem. It is possible to create more difficult puzzle with less restriction on valid Portia puzzles. When you solve different and more difficult Logical puzzle your overall understanding of the puzzle increases and you overall logical reasoning increases [19].

5.3.1 Valid Statements

In Portia puzzles the statements on the caskets are key to be able to solve the puzzle. In future work it might be possible to add new valid statements. These statements could both be new logical statements like *The gold casket contains one true statement.* One could also add statements that are logically equivalent to already valid statements like *The portrait is in the gold or silver casket.* This statement is logically equivalent to *The portrait is not in the Lead casket.* Adding extra valid statements would increase the number of Portia puzzles, which in turn allows you to create more complex Portia puzzles. Our program can be expanded to allow for more valid statements. This can be done by adding theses statements to the regex parser, adding a new function that is able to convert this statement into CNF, and adding this statement and all variations into the valid statements in the generator. The hardest part of adding new statements is that all puzzles still need to contain a unique solution. Any statement that causes ambiguity or solving loops are not solvable using our toolkit.

5.3.2 Number of Caskets

We gave a concrete definition for Portia-n-m in Definition 2. It could be possible to change this definition to allow a different number of statements per casket. This could allow you to create Portia puzzles that make better use of the positional properties of some statements like *The statements on this casket are true.*

We currently restrict the Portia puzzle to always have exactly three caskets. It is possible to extend this to any number of caskets. Our program can be expanded to allow for any number of caskets. You would need to change the natural language to also mention this number in the first line and add this new caskets statements to the end as n newlines. In the CNF formula the only thing that would change is the first section allowing only one casket to be true. Which is now hard code but would need a more general implementation.

Chapter 6

Conclusions

In this thesis, we aimed to answer the research question. Is it possible to formalize the structure of Portia puzzles in such a way that Portia puzzles can be generated and solved automatically? To answer this research question we split it up into three smaller sub questions.

How can you formalize Portia puzzles? We solved this sub question by giving the formal definition of Portia-n-m. We also give a way to represent Portia-n-m in natural language.

The second sub question is, How can you generate Portia puzzles. We developed a by hand construction algorithm to create valid Portia-n-m puzzles, validated our puzzles using both SAT and logic solver, and contributed several original puzzles not found in existing literature. We expanded the set of available Portia puzzles by giving examples with three and four statements per casket. We also gave a random Portia puzzle generator that can create even larger puzzles. The largest puzzle we were able to create is a Portia puzzle with 10 statements per casket. A puzzle with 11 statements is currently not possible with our implementation of Portia puzzles due to the total amount of valid statements in our implementation. The by hand constructing method works best for a smaller number of statements per casket. Whereas the random generation algorithm can be used for the larger Portia puzzles.

The third sub question is, How can you solve Portia puzzles. We presented a method for solving Portia puzzles using both logic-based reasoning and SAT solvers. We tested both these methods on a test set, both these methods gave the same solutions which gives confidence that both methods are correct.

The answers to the three sub questions answers the research question, Is it possible to formalize the structure of Portia puzzles in such a way that Portia puzzles can be generated and solved automatically? We conclude that it is possible to generate and solve Portia puzzles. Our toolkit enables the generation and solving of Portia puzzle without manual intervention.

This toolkit can be found on GitHub [13]. Using this toolkit someone can generate puzzle by hand and have the toolkit solve it or have the toolkit generate and solve a puzzle automatically.

Bibliography

[1] Gilles Audemard and Laurent Simon. On the Glucose SAT solver. *International Journal on Artificial Intelligence Tools (IJAIT)*, 27(1):1–25, 2018.

[2] Omid Bakhshandeh and James Allen. Semantic framework for comparison structures in natural language. In Lluís Màrquez, Chris Callison-Burch, and Jian Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 993–1002, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[3] Denis Berthier. Pattern-based constraint satisfaction and logic puzzles. Introduction. In Lulu.com, editor, *Pattern-based constraint satisfaction and logic puzzles*, page 480. Lulu.com, November 2012.

[4] Koen Claessen, Niklas Een, Mary Sheeran, Niklas Sörensson, Alexey Voronov, and Knut Åkesson. SAT-Solving in Practice, with a Tutorial Example from Supervisory Control. *Discrete Event Dynamic Systems*, 19(4):495–524, December 2009.

[5] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, 27(3):326–327, 1995.

[6] Hans Ditmarsch and Ji Ruan. Model checking logic puzzles, 11 2007.

[7] dmackinnon1. `https://www.mathrecreation.com/2017/12/constructing-portias-caskets.html`. [last accessed 17-17-2025].

[8] 3.13.2 Documentation. `https://docs.python.org/3/library/re.html`. [last accessed 17-17-2025].

[9] Elgun Jabrayilzade and Selma Tekir. LGPSolver - solving logic grid puzzles automatically. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1118–1123, Online, November 2020. Association for Computational Linguistics.

[10] Laura Kolijn. Generating and solving skyscrapers puzzles using a sat solver. *Radboud Bachlor Thesis*, 2022.

[11] Inês Lynce and Joël Ouaknine. Sudoku as a SAT problem. In *International Symposium on Artificial Intelligence and Mathematics, AI&Math 2006, Fort Lauderdale, Florida, USA, January 4-6, 2006*, 2006.

[12] Itsuki Maeda and Yasuhiro Inoue. Mathematical definition and systematization of puzzle rules, 2025.

[13] GitHub repository of codebase. `https://github.com/NoahRomeijnders/Portia-Puzzles`. [last accessed 17-17-2025].

[14] Joshua Erlangga Sakti, Muhammad Arzaki, and Gia Septiana Wulandari. Modeling path puzzles as sat problems and how to solve them. In Dieky Adzkiya and Kistosil Fahim, editors, *Applied and Computational Mathematics*, pages 227–245, Singapore, 2024. Springer Nature Singapore.

[15] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[16] Raymond Smullyan. *What Is the Name of This Book?* Prentice Hall, 1 edition, May 1978.

[17] `https://satcompetition.github.io`. [last accessed 17-17-2025].

[18] Nick van Oers. Pattern as an sat problem. *Radboud Bachlor Thesis*, 2024.

[19] Ting-Sheng Weng and Wenbing Zhao. Enhancing problem-solving ability through a puzzle-type logical thinking game. *Sci. Program.*, 2022, January 2022.