Generating Magical Performances with Constraint Programming

Guilherme de Azevedo Silveira
Alura, São Paulo, Brazil

Abstract

Professional magicians employ the use of interesting properties of a deck of cards to create magical effects. These properties were traditionally discovered through trial and error, the application of heuristics or analytical proofs. We discuss the limitations of relying on humans for such methods and present how professional magicians can use constraint programming as a computer-aided design tool to search for desired properties in a deck of cards. Furthermore, we implement a solution in Python making use of generative magic to design a new effect, demonstrating how this process broadens the level of freedom a magician can decree to their volunteers while retaining control of the outcomes of the magic. Finally, we demonstrate the model can be easily adapted to multiple languages.

2012 ACM Subject Classification Applied computing → Computer-aided design; Theory of computation → Constraint and logic programming

Keywords and phrases Constraint, generative design, computer aided design, constraint programming, generative magic, magical performance

Digital Object Identifier 10.4230/LIPIcs.CP.2021.10

Category Short Paper

Supplementary Material *Software*: `https://doi.org/10.5281/zenodo.5148915` [7]
Audiovisual: `https://doi.org/10.5281/zenodo.5148882` [6]

Acknowledgements We thank Daniela Mikyung Song for assistance providing the card designs and illustrations.

1 Introduction

A central problem in the performing arts of magic concerns designing new effects which are easily reproduced while complex enough so they are not easily figured out by spectators. Magicians traditionally use trial-and-error procedures that take time and are limited to only specific situations such as the creation process behind *Poker Night at the Improv* [14]. Others create heuristics such as the *System for Arranging Cards for Any Spelling Combination* [16], but heuristics might not work under different circumstances.

Mathematically inclined magicians publish proofs of properties on a deck of cards, such as the properties of a *Faro Shuffle* [10, 19, 21, 20]. Programmers create closed source code exploring possibilities on a memorized deck as in the *Poker Formulas* [14].

Even after an effect is published, it might not be replicated by magicians who perform them in other languages since many card effects use characteristics from their own cultures. One such example is the language bias present in many spellings of card effects, such as the value and suit in English which are fundamental parts of the *Spelling a Card* [16]. Other effects use double meanings of words such as Jacks or clubs. Memorized decks such as the *Aronson Stack* [1] are built around card spelling characteristics from the English language.

Moreover, nationality bias diminishes the reach of magical effects beyond the creator's cultural bubble. Poker is a central theme in many magical effects [23, 17, 11, 25, 15], although many countries have their own games [22] which better represent their identities.

© Guilherme de Azevedo Silveira;
licensed under Creative Commons License CC-BY 4.0

Therefore, designing effects based on the properties of a deck of cards is traditionally a time-consuming process, limited by the cultural aspects, biases and experience of a creator.

Mentalism

In *mentalism*, a volunteer makes n choices and the magician reveals a matching prediction [3]. While some effects allow for free choices, others will force the volunteer to a predetermined selection. One such example is the *P.A.T.E.O. force* [18]. The magician displays n items and, along with the volunteer, take turns removing items one by one. The remaining item is always one that the magician determined ahead of time. For the magic inclined, the method is further explained in Appendix A.

There are two hints to the volunteers that the magician is guiding them to choose a predetermined option. First, the magician is always involved in the process of removing an item, even if it is the volunteer's turn. Second, the magician makes the final choice. In this example and in the most common styles of forces, volunteers might perceive that they were forced to make a choice, thereby ruining the entire sensation of the uncontrolled environment the magician is trying to build.

This motivates us to ask whether there are methods to provide free choices to volunteers while retaining control on the final result of the effect.

Our contribution

By defining and limiting the parameter space of the choices of a volunteer, we demonstrate the application of constraint programming to control and infer results from random selections made by such a person. Python code is provided to replicate these findings [7] and a framework [8] was open sourced that can be used to design and generate magic under these premises.

Outline

This paper is organized as follows: Section 1 presents the background information necessary to understand the limitations of traditional creative magical methods. Section 2, then, examines how some magical effects can be generated through constraint programming; following which Section 3 briefly goes through the sample code that designs and creates such effects. Finally, Section 4 reports on the results achieved in performing these generative magic effects and discusses their real life usability before conclusions are presented in Section 5.

2 Controlling outcomes through constraint programming

Several spelling effects are described in the literature, where the magician deals a card as the chosen card name is spelled. A simple version of the spelling of a freely chosen card comprises of the following procedure.

The magician removes one blue and one red case from their pocket and places them on a table. The magician points to the blue case and announces that it contains one prediction. Next, they open the red one and spreads the 52 cards face down over the table, asking a volunteer to remove one card from the spread while hiding it from the magician.

The magician puts away both the remaining 51 cards and the red deck case back into their pocket. From the spectators' perspective, from this moment on, there is nothing the magician can do because both the prediction and the volunteer's free choice have already been made. The volunteer then reveals the chosen card was, for instance, the five of spades.

The performer proceeds to spell the name of the mentioned card, dealing one card from the top of the blue deck for each letter. Since the "five of spades" is spelled with 12 letters, the magician deals 12 cards and reveals the 13th card to be the five of spades. Spreading the other 51 cards face up, the magician reminds the volunteer that they could have chosen any card, yet, they both chose the same five of spades.

This effect consists of two parts, the first one is a free selection from the volunteer which turns out to be a forced card. To achieve this result, one can make use of a force red deck. One of such decks is a one-way deck consisting of 52 cards having the same red back and the same face, in this example, the five of spades. This self-working effect [12] is easy to perform because it only requires the preparation of the blue deck by positioning its five of spades in the 13th position.

Convincers and issues

To make the effect stronger, the magician can use false shuffles on both decks, such as the *Mead and Kennedy false shuffle* [5]. False shuffles temporarily displace some of the cards but end the movement with all cards in their original order prior to the shuffle.

They can follow it with false cuts, which do not change the deck order.

A third method consists in placing the five of spades at position 7 and execute controlled shuffles, such as the *Out-Faro Shuffle* [19] which brings the card to the 13th position. The Faro is described in Appendix A

Even the use of stronger convincers might not be enough for this effect as currently presented, since the magician cannot hand the red cards to the volunteers for further inspection. Depending on the force deck in use, it cannot even be spread face up.

Choices, control and knowledge

While the volunteer made a free choice in the previous effect, the magician controlled its result by giving them no other option but the five of spades. The magician decided beforehand which card would be chosen and placed it in its expected position into the deck.

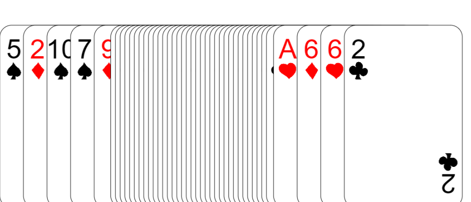
The selections made by the volunteer can be understood as parameters to the magic effect. Parameters can be randomly chosen from their own parameter space. In the example given, there is one [1,52] space, making it a 1-parameter magic effect.

The question raised is whether magical methods exist that allow volunteers to make N random choices over a N-dimensional parameter space while handing them real control over the card sequence and, yet, allow the magician to force the result.

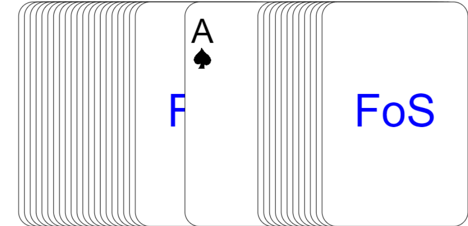
In this paper, we aim to confirm that it is indeed possible. A magician can make correct predictions about the outcome of N random choices made by a volunteer based on a set of items such as a 52 card deck by controlling other variables through constraint programming.

3 Constraining for freedom

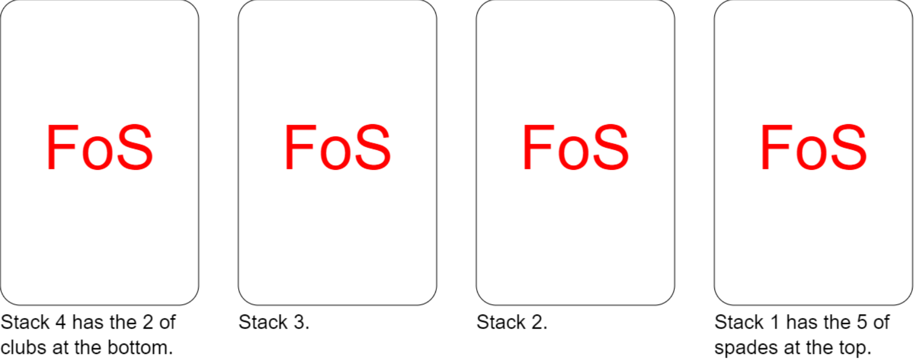
Our desired effect, *Freedom of Spelling*, consists of attributing real freedom over the parameters the volunteer will choose. The magician lays a blue and red case each on the table. The red cards are removed from the deck and fanned, revealing 52 different examinable cards as in figure 1a. They are spread face down on the table. The volunteer chooses 3 cut points resulting in 4 packets as in figure 1c, and decides a permutation that defines the sequence in which the packets will be put back together.



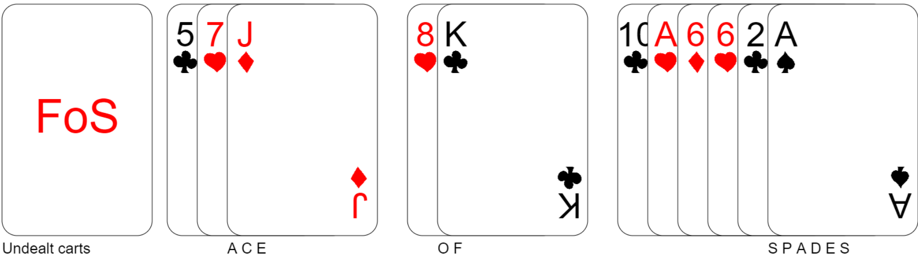
(a) Original sequence of the red deck.



(b) Revealing the ace of spades from the blue deck.



(c) Four piles after the selections of the volunteer.



(d) Revealing the ace of spades as it was spelled.

Figure 1 Performance of the Freedom of Spelling.

The volunteer has therefore 6 free choices and real control over the resulting card sequence. The magician emphasizes two points. First, the volunteer has made free choices. Second, there are 52! different combinations of a deck of cards. The magician proceeds to reveal one single card turned face down on the blue deck, for example, the ace of spades as in figure 1b. Spelling the “ace of spades”, the card is at the expected 12th position in the red deck which was controlled by the volunteer as in figure 1d.

There are two parts to this effect. In the first part, the volunteer freely makes 6 choices from a 6-dimensional parameter space. The resulting card sequence is one out of many possibilities. The volunteers are fooled to believe that there are 52! possible sequences that they might generate in this process while the magician has never explicitly said so. The two phrases are disconnected but the volunteers do not perceive it due to its misdirection and their misunderstanding of the possibilities generated by a 6-dimensional space parameter.

The first cut is made anywhere between position 2 and 49. The second cut between the first one and 50, the third one between the second one and 51. Finally the volunteer chooses any of the 24 permutations of the four stacks to gather the deck back together in one stack, which gives only a subset of all possible card sequence combinations, one in the order of 2 million possibilities.

The magician needs to know what position the volunteer has cut to. They can count the position by spreading the cards and asking the volunteer where to cut. Another method is to use number markings on the back of the cards as in *Card Control by the Numbers* [24].

The second part of the effect is to reveal that there is a card in its expected position. If the ace of spades is in the 12th position, the magician proceeds to use an index method to reveal the card as a supposed prediction. The method described earlier, the *Invisible Deck* [2], would display the ace of spades as the single card face down.

The question we are left with is will there always be at least one card in the expected position no matter the choice of parameters?

An unexpected way to model this problem aids in its solution. The magician has control of two aspects of the effect: the starting deck sequence and the card to be revealed. The first one is a set of 52 variables that start with no constraint. The card to be revealed should be defined by the parameters chosen by the volunteer during the live performance.

Because the first part of the effect defines 6 parameters and a sequence of array operations that are performed on the stack of cards, one can create a set of constraints that are required in order to guarantee the existence of one card in its expected position at the end of the process. In order to achieve it, we define the 6-dimensional parameter spaces. The 3 cut points are defined in closed intervals, and the final sequence is defined by the first 3 values of a permutation.

(a) $cut_1 \in [2, 49]$, $cut_2 \in [cut_1, 50]$, $cut_3 \in [cut_2, 51]$
(b) $sequence \in S(\{1, 2, 3, 4\})$

The second step is to extract the length of each card's name in the language that the effect will be performed. For instance, the ace of clubs has the length 10 while the king of diamonds has the length 14.

Given the 6 parameters and starting with a deck numbered $[1, 2, \ldots, 52]$, one can simulate the cuts and deck rebuilding, obtaining the final deck order. For example, cutting to the 10th, 20th and 30th card gives $cut_1 = 10$, $cut_2 = 20$ and $cut_3 = 30$. Using the permutation $[3, 1, 2, 4]$ results in the sequence 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, . . . , 20, 31, 32, 33, 34, . . . , 51, 52.

Note that the card that ends in the 11th position is the 1, while the 2 ends at position 12. If a card spelled with 10 letters begins at position 1 it would end at the expected 11th position.

One can now define a constraint that says the ace of clubs must start at position 1 to end at position 11. Another constraint requires that the king of spades starts at position 3. But those constraints do not need to hold at the same time, only one of the 52 constraints need to be satisfied as in listing 1.

Unfortunately, this is not yet enough to generate a magic effect. Satisfying one of such constraints gives us a set of starting deck position rules that work only for this specific point in the parameter space. The code must explore the entire parameter space, generating a set of 2 million constraints, each one consisting of an Or clause with 52 constraints.

Finally, adding the requirement that all variables are non-repeating integers in the space [1,52], a solver implementation can be used to check for satisfaction and, if possible, a unique solution that works for any set of parameters the volunteer chooses.

Listing 1 List of example constraints which must have at least one satisfied.

```
ace of clubs starts at 1 or
two of clubs starts at 1 or
...
king of spades starts at 3
```

Listing 2 A function that simulates the cuts and joins.

```
def simulate(cuts: List[int], deck: List[int]):
    stacks = np.array([range(cuts[0]),
                       range(cuts[0], cuts[1]),
                       range(cuts[1], cuts[2]),
                       range(cuts[2], 53)])
    return np.concatenate(stacks[[*deck]])
```

If a solution exists, the performer can use that specific order for the starting red deck, perform false shuffles and cuts. The volunteer makes their 6 choices. The performer looks up a table, as in *Poker Formulas* [14], or is hinted by a computer on which card is at the expected position in the red deck. The magician proceeds to reveal that card face down in the blue deck and finally spells the card in the correct position in the red one.

4 The Z3Solver solution

Starting with a deck in any order, one needs to simulate the card movements by slicing and joining arrays. Listing 2 creates a sequence and decides where each card in the original order of 1 to 52 ends up.

For example, a number 15 in the second position of the returned NumPy [13] array means that the 15th card from the original deck ends up at the 2nd position of the deck.

Using Z3Solver [9], a SMT solver, 52 integer variables are created representing the starting position of their respective cards as shown in listing 3. The first 13 variables stand for the ace, 2, 3, ..., 10, jack, queen and king of clubs. The next 39 variables represent the same cards in the suits of hearts, spades and diamonds.

Every card must have a constraint limiting its position to [1,52] and no two cards can occupy the same starting position.

The next step to constrain the original deck order to the requirements of a given set of 6 parameters is to simulate the card movements with a sample deck. Then, use its output to generate the required constraints as in listing 4.

The 52 conditions can be generated by going over each card extracting its name and length. By using the number that finishes at the expected position in the deck array, we define where such a card should be placed at the beginning of the effect, as in listing 5.

This allows the exploration of a single point in the parameter space, therefore, it is required to explore the 6-dimensional discrete parameter space, invoking *freedom_of_spelling* and generating a new rule for each execution as in listing 6.

Listing 3 Defining all 52 card variables.

```
names = map(retrieve_card_name, range(1, 53))
all_vars = list(map(z3.Int, names))
```

Listing 4 Simulates the slices, joins and return the proper constraints.

```
def freedom_of_spelling(all_vars:List[z3.Int], cuts:List[int],
                        sequence: List[int]) -> z3.Or:
    deck = simulate(cuts, sequence)
    return spelling_rules(all_vars, deck)
```

Listing 5 Card rules.

```
rules = []
for card in range(1, 53):
    name = retrieve_card_name(card)
    length = len(name.replace("␣",""))
    original_position = deck[length]
    rules.append(all_vars[card-1] == original_position)
return z3.Or(rules)
```

5 Results

The Z3Solver is unable to find a solution in less than 24 hours of runtime. The following optimizations and variations can be implemented to make its runtime faster and this method of magic creation more approachable to magicians without many resources.

Redesigning for a solution: multiple outs

The performer has only *one out* so far. For example, the "ace of spades" must be at position 12 so the magician reveals it when spelling the last letter. Such constraint can be relaxed by allowing the performer *two outs*. if the "ace of spades" is at position 13, the reveal is made after the complete spelling. The volunteers are never aware of the two possible outcomes. The relaxed constraint says that a card with n letters can be at position n or $n + 1$.

Optimizing

All card names in English have length between 10 and 15. Therefore, if different starting parameters end up with the same cards in these positions, the redundant ones are removed.

Also, although all cuts are possible, during performances, this is not true. Magicians know that volunteers do not make their cut on the first few cards even when given a free choice. Therefore, limiting the parameter space as follows achieves the same magical effect.

(a) $cut_1 \in [7, 21]$
(b) $cut_2 \in [max(cut_1, 12), 26]$
(c) $cut_3 \in [max(cut_2, 25), 39]$
(d) $sequence \in S(\{1, 2, 3, 4\})$

Listing 6 Exploring the 6-dimensional parameter space.

```
for cut1 in range(2, 49):
    for cut2 in range(cut1, 50):
        for cut3 in range(cut2, 51):
            for sequence in permutations(range(4)):
                cuts = (cut1, cut2, cut3)
                rule = freedom_of_spelling(all_vars, cuts, sequence)
                rules.append(rule)
```

Table 1 Stack order for *Freedom of Spelling in English*, from the deck top to its face.

1	5 of hearts	14	8 of clubs	27	7 of diamonds	40	8 of spades
2	2 of diamonds	15	J of spades	28	4 of diamonds	41	K of hearts
3	10 of spades	16	3 of diamonds	29	J of hearts	42	6 of spades
4	7 of spades	17	K of spades	30	J of diamonds	43	J of clubs
5	9 of diamonds	18	A of clubs	31	9 of hearts	44	7 of hearts
6	7 of clubs	19	4 of spades	32	K of diamonds	45	3 of hearts
7	9 of clubs	20	5 of clubs	33	5 of spades	46	10 of hearts
8	8 of hearts	21	9 of spades	34	5 of diamonds	47	Q of spades
9	6 of clubs	22	Q of hearts	35	3 of clubs	48	10 of clubs
10	A of diamonds	23	10 of diamonds	36	4 of clubs	49	A of hearts
11	Q of clubs	24	4 of hearts	37	8 of diamonds	50	6 of diamonds
12	K of clubs	25	2 of hearts	38	3 of spades	51	6 of hearts
13	A of spades	26	2 of spades	39	Q of diamonds	52	2 of clubs

After optimizing, 2,049 constraints for English are created in 8 minutes and analyzed, generating one of many sequences of the deck that satisfies the desired properties.

The entire source code was released [7] and can be found in Appendix B. A new project was released as a library [8], allowing magicians to explore methods, create effects and share contributions to advance the field of generative magic.

Preshow and performance

The resulting stack order, *Freedom of Spelling in English*, is presented in table 1. Other solutions exist for card name length tables using different languages.

During the performance, the simulation function is run once to obtain the matching card for the set of parameters chosen by the volunteer.

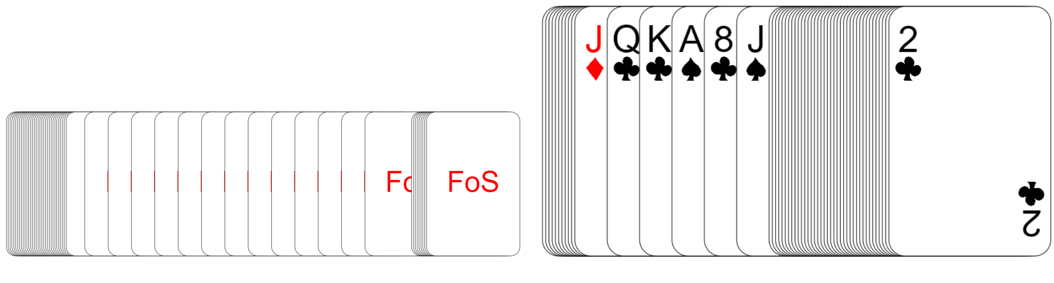
The effect was performed online in English, Korean and Portuguese by a professional magician. In a close up presentation it is preferred to use low tech - such a clicker - to input the position of the cuts. A smartwatch or hidden thumper can notify the magician which the card is at its expected position. These technologies are already in use in close up magic performances.

As a performance example, figure 1a shows the stacked deck from table 1. A volunteer makes the first cut as in figure 2a.

Supposing the cuts are made at positions 10, 20 and 30. When the volunteer rebuilds the stack using the sequence [3, 2, 1, 4], the resulting 10th to 15th cards can be seen in figure 2b: the jack of diamonds, queen of clubs, king of clubs, ace of spades, eight of clubs and jack of spades. In this case, the king of clubs is at position 12, its length plus 1. Therefore, in a second deck, the magician displays a single card face down, the king of clubs. They spell "king of clubs" while dealing 12 cards, revealing the king of clubs as in 2c. The entire performance with the explanation was made available [6].

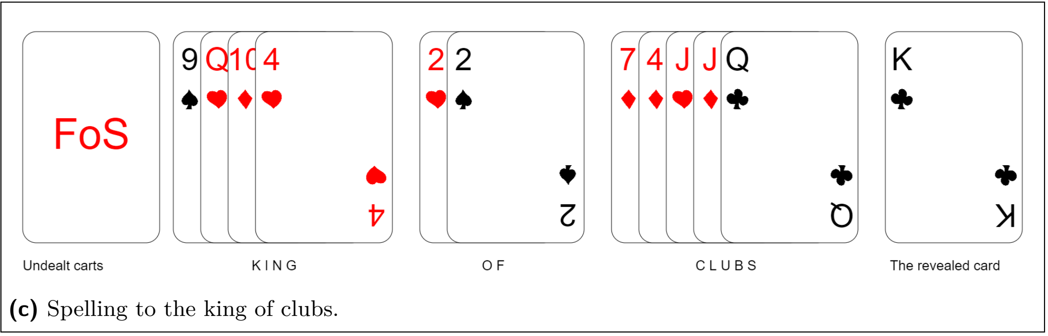
6 Conclusion and further research

We presented a novel problem formulation to the design process of new effects in magic. Our implementation proves that new magical effects can be devised and implemented through the use of computer-aided design (CAD) frameworks. We devised and applied one effect using constraint programming, giving the volunteers more freedom while keeping control of the results with the magician.



(a) Controlling the first cut of a volunteer.

(b) Peeking at the rebuilt deck.



(c) Spelling to the king of clubs.

Figure 2 Simulation of a performance with parameters $[10, 20, 30, 3, 2, 1, 4]$.

Because the code is open source, anyone can run it using a different language, therefore, limiting some of the unintentional biases and boundaries a magical effect has due to the identity and experience of its creator.

Further research can be done optimizing the model, using other solvers and trying to remove the constraints from the parameter space. Other effects can be generated such as allowing a volunteer to choose the language to be used only after the deck has been handled by the performer.

Generative magic can be explored with other areas of mathematics and computer science. Its ultimate research challenge is to search and catalog in programming terms the existing effects in magic literature so that an engine can co-design new ones.

References

1 Simon Aronson. *A Stack To Remember*. Self-published, 1979.
2 J. B. Bobo. *WATCH THIS ONE!* Lloyd E. Jones, 1947.
3 Robert Cassidy. *The Art of Mentalism*. Collectors' Workshop, 1984.
4 Michael Close. *Workers Number 5*. Self-published, 1996.
5 Michael Close. *Closely Guarded Secrets*. MichaelClose.com, 2 edition, 2004.
6 Guilherme de Azevedo Silveira. Freedom of spelling demonstration, 2021. `doi:10.5281/zenodo.5148882`.
7 Guilherme de Azevedo Silveira. Freedom of spelling source code, July 2021. `doi:10.5281/zenodo.5148915`.
8 Guilherme de Azevedo Silveira. Generative magic, 2021. URL: `https://github.com/guilhermesilveira/generativemagic/`.

9 Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, C. R. Ramakrishnan, and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963, pages 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. `doi:10.1007/978-3-540-78800-3_24`.

10 Persi Diaconis, R.L Graham, and William M Kantor. The mathematics of perfect shuffles. *Advances in Applied Mathematics*, 4(2):175–196, 1983. `doi:10.1016/0196-8858(83)90009-X`.

11 Karl Fulves. *Hocus Poker*. Self-published, 1982.

12 Glenn Gravatt. *Encyclopedia of Self-working Card Tricks*. Quality Magic Company, 1936.

13 Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. `doi:10.1038/s41586-020-2649-2`.

14 Pit Hartling. *In Order To Amaze*. Self-published, 2016.

15 Jean Hugard and Frederick Braue. *The royal road to card magic*. Farber and Farber, London, 1979. OCLC: 25036735.

16 Jean Hugard, John J. Crimmins, and Glenn G. Gravatt. *Encyclopedia of card tricks*. Dover Publications, New York, 1974.

17 Edward Marlo. *The Ten Hand Poker Stack*. Self-published, 1974.

18 Hugh Miller. *Baker's Bonanza*. Supreme Magic Publication, 2 edition, 1978.

19 Stephen Minch. *The Collected Works of Alex Elmsley*, volume 2. L & L Publishing, 1994.

20 S. Brent Morris. The basic mathematics of the faro shuffle. *Pi Mu Epsilon Journal*, 6(2):85–92, 1975. URL: `http://www.jstor.org/stable/24345166`.

21 S.Brent Morris and Robert E. Hartwig. The generalized faro shuffle. *Discrete Mathematics*, 15(4):333–346, 1976. `doi:10.1016/0012-365X(76)90047-9`.

22 Nike Arts. *Enciclopedia de los juegos de cartas*. RobinBook, Barcelona, 1999. OCLC: 807845266.

23 Darwin Ortiz. *Darwin Ortiz on casino gambling: the complete guide to playing and winning*. Dodd, Mead & Co, New York, 1st ed edition, 1986.

24 McCabe Pete and Michael Close. *The PM Card System*. MichaelClose.com, 2019.

25 David Regal. *Close-up & personal*. Hermetic Press, Seattle, Wash., 1999. OCLC: 45272617.

A Mentioned effects

Magic is an uncommon theme to the field of constraint programming. Therefore, this appendix further describes a few of the effects mentioned earlier.

P.A.T.E.O. force

During a dinner event, the performer writes “bottle” in a piece of paper and leaves it folded over the table, in plain sight. Nobody but the performer knows what is written as this is the magician's prediction. A volunteer is chosen and the performer selects and names 5 items for them to play with, such as a napkin, glass, bottle, fork and knife.

The magician points to two of those items. The volunteer is asked to select one amongst the two to remove. With four elements left, it is the volunteer's turn to point to two items. The magician selects one amongst the two and removes it. This process is repeated until there is only one element, which will always be the prediction element – in this case, the bottle.

For this effect to work, the magician must always select two items which do not include the prediction. During the volunteer's turn, if they point to two items that do not include the prediction, the magician can remove any element. If the volunteer points to one item and the prediction item, the magician must remove the former. Because the magician goes first to show how it works, the number of items must be odd for the effect to work.

The Faro shuffle

Although called a shuffle, the Faro is a movement which controls the entire deck. A typical deck of 52 cards is split into two stacks of 26 cards. Each stack is interwoven resulting in a single stack where all even cards come from the original top and odd ones from the original bottom. The result might be the inverse according to which card is the first one to interweave.

The magician must practice the precise cut and interweave movements as to pretend to be doing an uncontrolled shuffle, while in reality controlling the position of the cards.

The simplest usage of a series of Faro shuffles is to force one card into a specific target position.

Stacked decks

Stacked decks are either partially or completely memorized and used throughout magic performances. The cards at position 10 to 15 in the Aronson Stack are the ace of clubs, ten of spades, five of hearts, two of diamonds, king of diamonds and seven of diamonds. All of them are at their exact spelling position when using the English language.

Invisible deck

The invisible deck is traditionally performed by first displaying a closed deck case. After asking a volunteer to name any card the magician opens the case and shows that there is only one card face down. It happens to be the card named by the volunteer.

The widely used Invisible Deck can not be examined after the effect is performed because its gimmick is easily perceived when the cards are manipulated. Michael Close's variation [4] does not use any gimmick and is fully examinable after the reversed card is revealed.

Invisible decks are used as finishers, to show a matching prediction. However, its usage in *Freedom of Spelling* is innovative as it forces an intermediate outcome.

B Source code

The source code for generating one *Freedom of Spelling* deck sequence can be split into generating the variables in listing 7 and generating the constraints and running the solver in listing 8.

Listing 7 Generating the deck sequence.

```
from itertools import permutations
from typing import List, Tuple
import numpy as np
import z3

def simulate(cuts: Tuple[int], deck: Tuple[int]):
    stacks = np.array([range(cuts[0]),
                       range(cuts[0], cuts[1]),
                       range(cuts[1], cuts[2]),
                       range(cuts[2], 53)])

    return np.concatenate(stacks[[*deck]])

VALUES = ["ace", "two", "three", "four", "five", "six", "seven",
          "eight", "nine", "ten", "jack", "queen", "king"]
SUITS = ["clubs", "hearts", "spades", "diamonds"]

def retrieve_card_name(position):
    value = (position - 1) % 13
    suit = (position - 1) // 13
    return VALUES[value] + "␣of␣" + SUITS[suit]

def rules_all_cards_on_deck(all_vars: List[z3.Int]) -> z3.And:
    rules = [z3.And([card >= 1, card <= 52]) for card in all_vars]
    return z3.And(rules)

def freedom_of_spelling(all_vars: List[z3.Int], cuts: Tuple[int],
                        sequence: Tuple[int]) -> z3.Or:
    deck = simulate(cuts, sequence)
    return spelling_rules(all_vars, deck)

def spelling_rules(all_vars: List[z3.Int], deck: List[int]) -> z3.Or:
    rules = []
    for card in range(1, 53):
        name = retrieve_card_name(card)
        length = len(name.replace("␣", ""))
        original_position = int(deck[length])
        var = all_vars[card - 1]
        rules.append(var == original_position)
        rules.append(var == original_position - 1)
    return z3.Or(rules)

rules = set()
names = map(retrieve_card_name, range(1, 53))
all_vars = list(map(z3.Int, names))
```

Listing 8 Generating all optimized constraints.

```
from tqdm import tqdm
from z3 import AtMost

for cut1 in tqdm(range(7, 21)):
    for cut2 in range(max(cut1, 12), 26):
        for cut3 in range(max(cut2, 25), 39):
            for sequence in permutations(range(4)):
                cuts = (cut1, cut2, cut3)
                rule = freedom_of_spelling(all_vars, cuts, sequence)
                rules.add(rule)

for position in range(1, 53):
    at_starting_point = [card == position for card in all_vars]
    rule = AtMost(*at_starting_point, 1)
    rules.add(rule)

rules.add(rules_all_cards_on_deck(all_vars))

print(len(rules))
print(z3.solve(rules))
```