

# Theory of Computation

## SAT Encoding

Andrea Gualandris  
Arianna Bianchi  
Elena Franchini  
Martino Giorgi

### 1 Folder structure

```
SAT-solver
├── solver.py
├── website
│   ├── main.py
│   ├── templates
│   └── static
│       ├── css
│       └── img
├── README.md
├── input.txt
├── requirements.txt
├── report
└── test_cases.txt
```

The solver function is implemented inside the file `solver.py`. The folder `website` contains the files to implement the interface with which we can start the application on port 3000.

## 2 Installation

All the following commands are supposed to run from the main folder SAT-solver.  
To install the dependencies:

```
pip install -r requirements.txt
```

To run the solver from the command line:

```
python3 solver.py input.txt
```

To run the application on the browser:

```
python3 website/main.py
```

To see the application, go to: <http://127.0.0.1:3000>.

## 3 Instructions

The SAT solver reads an input file which contains couple of clothes and colors, that the user wants to wear, expressed always in this form  $\langle cloth, color \rangle$ . An example could be:  $\langle skirt, red \rangle$ , While an example of file input could be:

```
skirt red
shirt green
jacket black
```

From the web application the clothes and colors are chosen through the interface showed in picture 1. Here it's possible to select a color for each cloth. It's not mandatory to select every cloth, you can chose the matches that you prefer the most.


Once you select the outfit, you can click on "Can I wear this outfit?" and a message appears saying if your outfit satisfies the constraints as in picture 2, or not as in picture 3.

## 4 Design and Implementation

We decided to implement the constraints as couples of clothes and colors since it's a simple and optimal solution. We chose these sets:


```
clothes = {shirt, pullover, jacket, trousers, skirt, shoes, flipFlops, hat, sunglasses}
colors = {white, black, blue, pink, brown, green, yellow, red, orange}
```

• Outfit Adviser •




Hat

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐




Sun Glasses

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐




Shirt

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐




Pullover

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐




Jacket

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐




Trousers

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐




Skirt

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐



Shoes

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐



Flip Flops

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

[Can I wear this outfit?](#)

Figure 1: Interface where you can decide how to wear

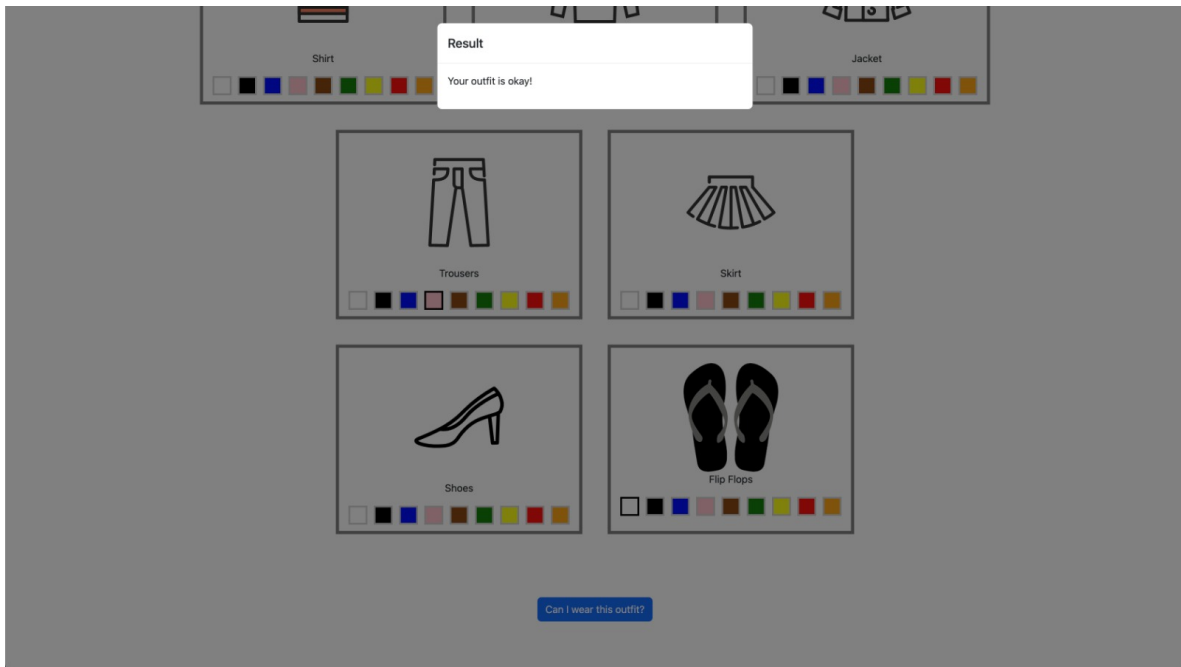


Figure 2: Message saying your outfit satisfies the constraints.

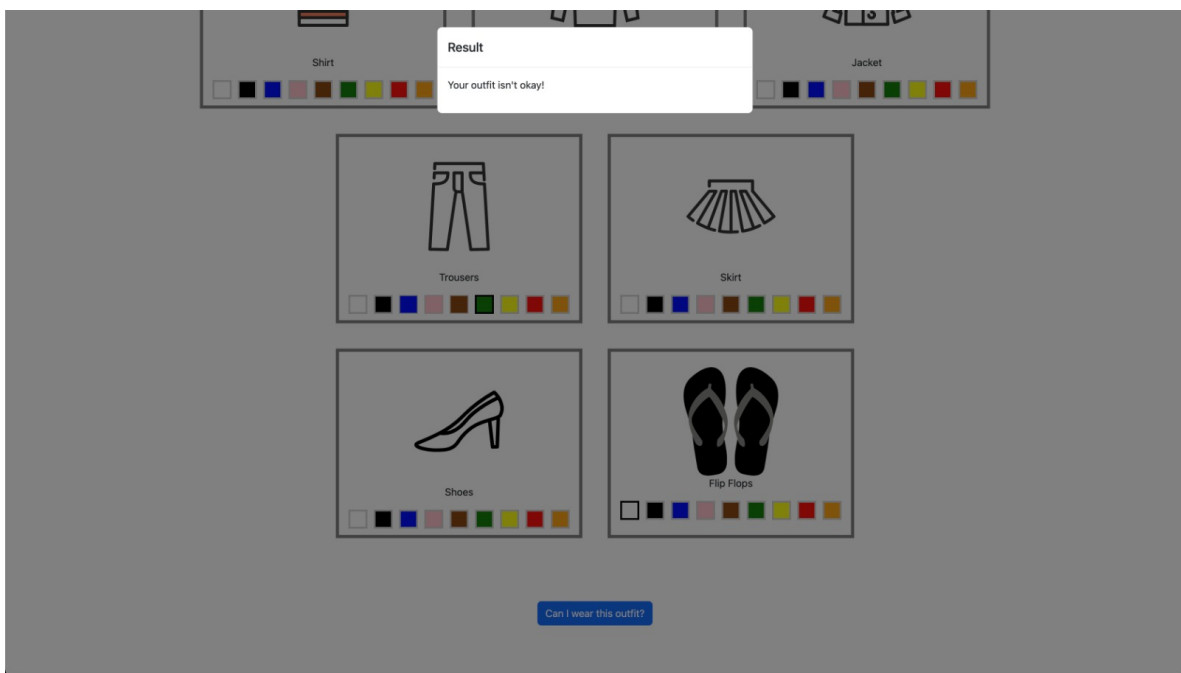


Figure 3: Message saying your outfit doesn't satisfy the constraints.

## 5 SAT solver

As first step, we created two dictionaries for all the clothes and all the colors relatively. We defined a Solver and we added all the constraints into the Solver. Next step, we pass the input file - with the clothes and colors that the user wants to wear - to the Solver, in order to store them with an AND gate. In this way the Solver already know if the constraints (cloth and color) are satisfied or not. Last step is to check if the solver is SAT which succeeds, or not SAT which fails.

The table of constrains we added, represented as boolean expression is the following:

Boolean expression for constraints
$\neg(\text{skirt} \wedge \text{trousers})$
$\neg(\text{shoes} \wedge \text{flipFlops})$
$\neg(\text{pullover} \wedge \text{jacket})$
$\neg(\text{pullover} \wedge \text{flipFlops})$
$(\text{pullover} \rightarrow \text{shirt})$
$(\text{jacket} \rightarrow \text{shirt})$
$(\text{jacket} \rightarrow (\text{shirt} \vee \text{trousers}))$
$((\text{flipFlops} \vee \text{sunglasses}) \rightarrow \text{hat})$
$\neg(\text{red} \wedge \text{orange})$
$\neg(\text{pink} \wedge \text{red})$
$\neg(\text{blue} \wedge \text{green})$
$\neg(\text{brown} \wedge \text{pink})$
$\neg(\text{pink} \wedge \text{green})$
$\neg(\text{green} \wedge \text{yellow})$
$\neg(\text{blue} \wedge \text{yellow})$
$\neg(\text{red} \wedge \text{green})$
$\neg(\text{brown} \wedge \text{black})$

We wrote the entire code for the Solver in Python, precisely in file Solver.py. To implement it we used z3-solver library, and pandas to read the input file.

## 6 Problems encountered

One problem we encountered was how to add the inputs given by the user with our constraints, but we solve it seeing that in the z3 library there is a method *add()* that permits automatically to add constraints (in our case, the inputs) to the solver.

## 7 Alternative implementations

At the beginning our idea was to implement manually the solver, but it would result in a more complex process and messy code. After some searches on Google, we found out the library z3-solver and we smoothly implemented it, with a clean and ordered code.