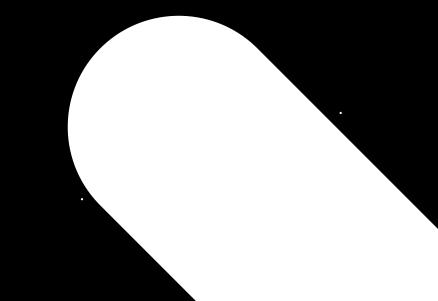
H T W I G N

Solve Dinner Planning with CSP

MSI - DIMA - Task 1B

Nadina Brodt, Johannes Brandenburger, Tobias Tögel github.com/johannesbrandenburger/htwg-msi-dima



Approach

- formulate dependencies
- build CNF (clauses)
- create own simple SAT algorithm
 - search for unit clauses ("A")
 - add them to result
 - eliminate them and their negations in the clauses
 - recursively call the function with additional unit clauses

Implementation

- formulate dependencies
- build CNF (clauses)
- create own simple SAT algorithm
 - search for unit clauses ("A")
 - add them to result
 - eliminate them and their negations in the clauses
 - recursively call the function with additional unit clauses

```
## Abbreviations
Mr H
Mrs W
Emma E
Georg G
Ivana I
## Constrains in CNF
(-H \text{ or } W) \qquad -> ["-H", "W"]
(I or G) -> ["I", "G"]
(W \text{ or } E) -> ["W", "E"]
(not W or not E) -> ["-W", "-
f=f or G) -> ["-E", "G"]
(-G or E) -> ["-G", "E"]
                 -> ["-I", "G"]
(-I \text{ or } G)
                 -> ["-I", "H"]
(-I \text{ or } H)
```

Implementation

- formulate dependencies
- build CNF (clauses)
- create own simple SAT algorithm
 - search for unit clauses ("A")
 - add them to result
 - eliminate them and their negations in the clauses
 - recursively call the function with additional unit clauses

```
def remove_unit_clauses(result, clauses):
   while True:
        unitClauses = []
        for clause in clauses:
            if len(clause) == 1 and clause[0] not in unitClauses:
               unitClauses.append(clause[0])
        if len(unitClauses) == 0:
            break
        for (i, clause) in enumerate(clauses):
            for unitClause in unitClauses:
                if unitClause in clause:
                    clauses[i] = None
        clauses = [clause for clause in clauses if clause is not None]
        for (i, clause) in enumerate(clauses):
            for unitClause in unitClauses:
                if neg(unitClause) in clause:
                    clauses[i].remove(neg(unitClause))
        for unitClause in unitClauses:
            if neg(unitClause) in result:
                return [], clauses
            if unitClause not in result:
                result.append(unitClause)
   return unique(result), clauses
```

Implementation

- formulate dependencies
- build CNF (clauses)
- create own simple SAT algorithm
 - search for unit clauses ("A")
 - add them to result
 - eliminate them and their negations in the clauses
 - recursively call the function with additional unit clauses

```
def simple_csp_solver(result: list[str], clauses: list[list[str]]):
    result, clauses = remove unit clauses(result, clauses)
    for clause in clauses:
        if len(clause) == 0:
            return []
    if len(clauses) == 0:
        return result
    literal = clauses[0][0]
    clauses 1 = clauses + [[literal]]
    clauses 2 = clauses + [[neg(literal)]]
    result_1 = simple_csp_solver(result, clauses_1)
    if len(result_1) > 0:
        return result 1
    return simple_csp_solver(result, clauses_2)
constrains = [...] # see markdown
should solution = ["-H", "-W", "-I", "G", "E"] # found by brute force
is solution = simple csp solver([], constrains)
print("should_solution: ", should_solution) # ['-H', '-W', '-I', 'G',
print("is solution: ", is solution) # ['-H', '-I', '-W', 'E', 'G']
assert(sort_array(should_solution) == sort_array(is_solution))
```

H T W I G N

Solve Dinner Planning with CSP

MSI - DIMA - Task 1B

Nadina Brodt, Johannes Brandenburger, Tobias Tögel github.com/johannesbrandenburger/htwg-msi-dima

