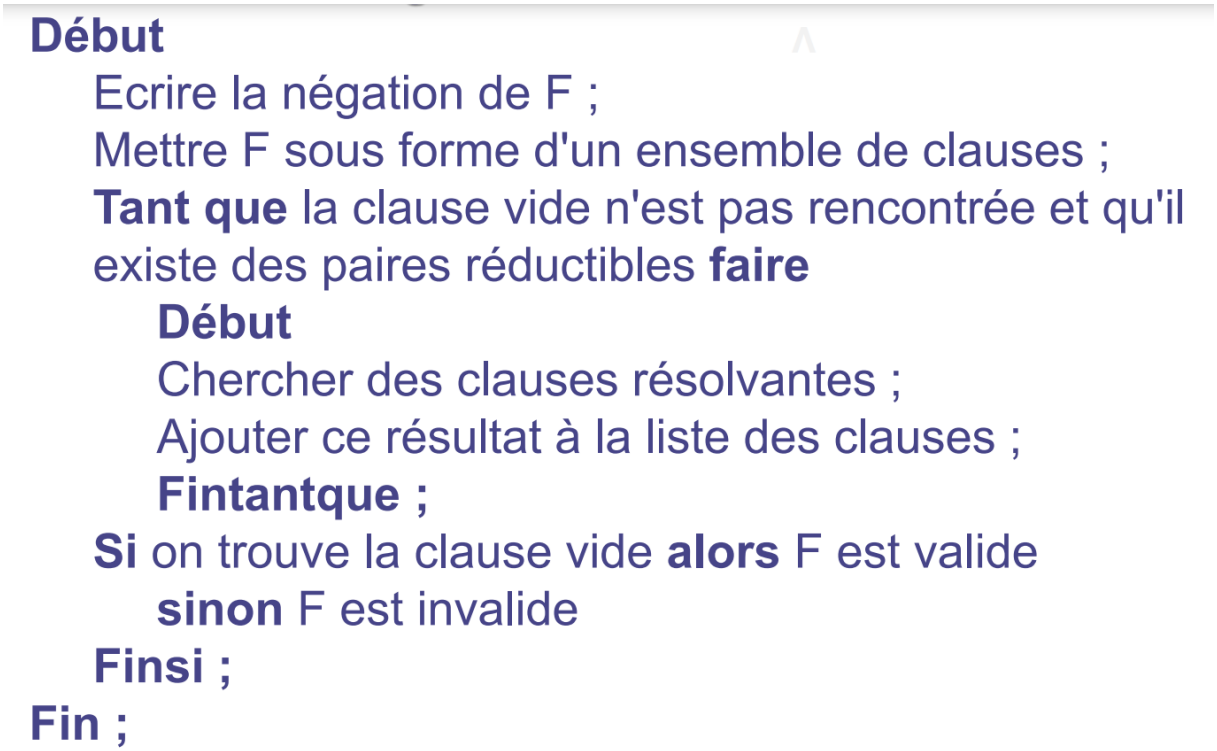
Algorithme de résolution

SAIED Romaysae

L'algorithme de résolution est une méthode puissante pour déterminer si une formule propositionnelle est valide ou invalide. Il fonctionne en exploitant la propriété de réfutabilité de la logique propositionnelle, qui stipule qu'une formule est invalide si et seulement si sa négation est satisfiable, on trouve si dessous un exemple de cet algorithme :



Le code si dessous est une implémentation Python de l'algorithme de résolution, une méthode utilisée pour prouver l'insatisfaisabilité d'un ensemble de clauses dans une formule logique propositionnelle. L'algorithme de résolution est une procédure de décision pour le problème de la satisfiabilité logique propositionnelle.

Code :

```
def parse_formula_input(input_string):
    clauses_match = re.findall(r'\{(.*?)\}', input_string)
    if not clauses_match:
        raise ValueError("Invalid input format. Please use the
            format {¬P v ¬Q v R, ¬R, P, ¬T v Q, T}")

    clauses = [re.split(r'\s*v\s*', clause) for clause in
        clauses_match[0].split(',')]

    processed_clauses = []
    for clause in clauses:
        processed_clause = []
        for literal in clause:
            if "¬" in literal:
                processed_clause.append(-ord(literal[1]) + ord
                    ("A") + 1)
            else:
                processed_clause.append(ord(literal[0]) - ord
                    ("A") + 1)
        processed_clauses.append(processed_clause)

    return processed_clauses

def negation(F):
    if isinstance(F[0], list):
        return [[-literal for literal in clause] for clause in
            F]
    else:
        return [-F]

def mettre_sous_forme_de_clauses(F):
    return [F] if isinstance(F[0], int) else F

```

```python
def chercher_clauses_resolvantes(clauses):
    result = []
    for i in range(len(clauses)):
        for j in range(i+1, len(clauses)):
            for literal_i in clauses[i]:
                for literal_j in clauses[j]:
                    if abs(literal_i) == abs(literal_j):
                        resolvant = [literal for literal in
                            clauses[i] if literal != literal_i]
                            + \
                                    [literal for literal in
                               clauses[j] if literal !=
                               literal_j]
                        if resolvant not in result:
                            result.append(resolvant)
    return result
```

```python
def resolution(F):
    neg_F = negation(F)
    clauses = mettre_sous_forme_de_clauses(neg_F)
    while True:
        new_resolvantes = chercher_clauses_resolvantes(clauses)
        if not new_resolvantes:
            return "F est valide"

        for resolvant in new_resolvantes:
            if all(lit not in clauses for lit in resolvant):
                clauses.append(resolvant)

        if [] in clauses:
            return "F est invalide"
```

```python
def test_resolution():
    formula_input = input("Enter the formula : ")

    try:
        formula = parse_formula_input(formula_input)
    except ValueError as e:
        print(e)
        return

    print("Formula:", formula)
    print("Result:", resolution(formula))
    print()
test_resolution()
```

Le code définit plusieurs fonctions :

1. **parse_formula_input** : Cette fonction prend en entrée une chaîne de caractères au format {¬P v ¬Q v R, ¬R, P, ¬T v Q, T} ou {P ∨ Q, ¬P ∨ R, ¬Q ∨ R} et renvoie une liste de clauses traitées, où chaque clause est représentée sous forme d'une liste d'entiers. Le premier entier de la liste est le numéro de la variable, et si l'entier est négatif, il représente la négation de la variable.
2. **negation** : Cette fonction prend une liste de clauses et renvoie la négation de la liste de clauses.
3. **mettre_sous_forme_de_clauses** : Cette fonction prend une liste de clauses et renvoie la liste de clauses au format attendu par l'algorithme de résolution.
4. **chercher_clauses_resolvantes** : Cette fonction prend une liste de clauses et renvoie une liste de clauses résolventes, qui sont obtenues en résolvant des paires de clauses dans la liste.
5. **resolution** : Cette fonction prend une liste de clauses et renvoie une chaîne de caractères indiquant si la liste de clauses est valide ou non. L'algorithme de résolution procède en recherchant des clauses résolventes et en les ajoutant à la liste de clauses jusqu'à ce qu'aucune nouvelle clause résolvente ne puisse être trouvée. Si la clause vide est obtenue, la liste de clauses est insatisfaisable, et la fonction renvoie "F est invalide". Sinon, si aucune nouvelle clause résolvente ne peut être trouvée, la liste de clauses est satisfaisable, et la fonction renvoie "F est valide".
6. **test_resolution** : Cette fonction ne prend aucun argument et teste l'algorithme de résolution sur une liste de clauses fournie par l'utilisateur.

Pour tester le code on utilise 2 formules :

1. Formule invalide :

```
Enter the formula : {¬P v ¬Q v R, ¬R, P, ¬T v Q, T}
Formula: [[-14, -15, 18], [-106], [-32], [-106, 17], [-32]]
Result: F est invalide
```

2. Formule valide :

```
Enter the formula : {¬P v ¬Q ,R, ¬P}
Formula: [[-14, -15], [18], [-106]]
Result: F est valide
```

La fonction parse_formula_input a traité cette entrée en une liste de clauses, qui a ensuite été transmise à la fonction resolution. La fonction resolution a déterminé que la liste de clauses est satisfaisable et a renvoyé "F est valide".