SAT Solver
COSE419, Spring 2024

Hakjoo Oh

Due: 4/5 23:59

Problem 1 *선언 논리식(Propositional formula)을 다음과 같이 OCaml 타입으로 정의할 수 있다.*

```
type var = string
type formula =
  | False
  | True
  | Var of var
  | Not of formula
  | And of formula * formula
  | Or of formula * formula
  | Imply of formula * formula
  | Iff of formula * formula
```

예를 들어 $(P \rightarrow Q) \wedge (P \wedge \neg Q)$는 다음과 같이 표현된다.

```
And (
  Imply (Var "P", Var "Q"),
  And (Var "P", Not (Var "Q"))
)
```

이번 과제의 목표는 `formula` 타입의 논리식이 주어졌을때 SAT/UNSAT 여부를 판단하는 함수 `solve`의 구현을 완성하는 것이다.

```
solve : formula -> bool
```

`solve`는 참이 될 수 있는(SAT) 논리식이 주어지면 *true*를 리턴한다. 논리식의 해가 존재하지 않는다면(UNSAT) `false`를 리턴한다. 예를 들어,

```
# solve (
  Or (
    And (Var "Q1", Var "Q2"),
    And (Var "R1", Var "R2")
  ));;
- : bool = true

# solve (
  And (
    Imply (Var "P", Var "Q"),
    And (Var "P", Not (Var "Q"))
  ));;
- : bool = false

# solve (Or (And (Var "P", Var "Q"), Var "R"));;
- : bool = true
```

함수 solve의 얼개는 다음과 같다.

```
let solve : formula -> bool
=fun f -> dpll (convert f)
```

1. 주어진 논리식을 "equisatisfiable CNF"로 변환하는 함수 convert를 작성하시오.

   ```
   convert : formula -> cnf
   ```

   타입 cnf는 다음과 같이 정의한다.

   ```
   type literal = bool * var (* false means negated *)
   type clause = literal list
   type cnf = clause list
   ```

   예를 들어, $P \wedge (\neg P \vee Q) \wedge (R \vee \neg Q \vee S)$는 다음과 같이 표현된다.

   ```
   [[(true, "P")];
    [(false, "P"); (true, "Q")];
    [(true, "R"); (false, "Q"); (true, "S")]]
   ```

   함수 convert의 동작 예시는 다음과 같다 (생성되는 임시 변수들의 이름은 다를 수 있다).

   ```
   # convert True;;
   - : cnf = []

   # convert False;;
   - : cnf = [[]]

   # convert (Imply (Var "P", Var "Q"));;
   - : cnf =
   [[(true, "P(P -> Q)")]; [(false, "P(P -> Q)"); (false, "P"); (true, "Q")];
    [(true, "P"); (true, "P(P -> Q)")]; [(false, "Q"); (true, "P(P -> Q)")]]

   # convert (Or (And (Var "P", Var "Q"), Var "R"));;
   - : cnf =
   [[(true, "P((P and Q) or R)")];
    [(false, "P((P and Q) or R)"); (true, "P(P and Q)"); (true, "R")];
    [(false, "P(P and Q)"); (true, "P((P and Q) or R)")];
    [(false, "R"); (true, "P((P and Q) or R)")];
    [(false, "P(P and Q)"); (true, "P")]; [(false, "P(P and Q)"); (true, "Q")];
    [(false, "P"); (false, "Q"); (true, "P(P and Q)")]]

   # convert (And (Imply (Var "P", Var "Q"), And (Var "P", Not (Var "Q"))));;
   - : cnf =
   [[(true, "P((P -> Q) and (P and (not Q)))")];
    [(false, "P((P -> Q) and (P and (not Q)))"); (true, "P(P -> Q)")];
    [(false, "P((P -> Q) and (P and (not Q)))"); (true, "P(P and (not Q))")];
    [(false, "P(P -> Q)"); (false, "P(P and (not Q))");
     (true, "P((P -> Q) and (P and (not Q)))")];
    [(false, "P(P -> Q)"); (false, "P"); (true, "Q")];
    [(true, "P"); (true, "P(P -> Q)")]; [(false, "Q"); (true, "P(P -> Q)")];
    [(false, "P(P and (not Q))"); (true, "P")];
    [(false, "P(P and (not Q))"); (true, "P(not Q)")];
    [(false, "P"); (false, "P(not Q)"); (true, "P(P and (not Q))")];
    [(false, "P(not Q)"); (false, "Q")]; [(true, "P(not Q)"); (true, "Q")]]
   ```

2. DPLL 알고리즘은 다음과 같이 구현할 수 있다. CNF로 표현된 주어진 논리식이 SAT인 경우 *true*를 리턴한다.

```
let rec dpll : cnf -> bool
=fun a ->
  let a = ple (bcp a) in
    if a = [] then true  (* empty conjunction = true *)
    else if List.mem [] a then false (* empty clause = false *)
    else
      let x = choose a in
        dpll (subst a false x) || dpll (subst a true x)
```

(a) 치환(Substitution) `subst`를 구현하시오.

```
subst : cnf -> bool -> var -> cnf
```

예를 들어,

```
# subst [[(true, "P")]] true "P";;
- : cnf = []

# subst [[(false, "P")]] true "P";;
- : cnf = [[]]

#  subst [[(false, "P"); (true, "Q")]] true "P";;
- : cnf = [[(true, "Q")]]

# subst [[(false, "P"); (true, "Q")]] false "P";;
- : cnf = []

# subst  [[(false, "P"); (true, "Q")];
              [(true, "R"); (false, "Q"); (true, "S")]] false "Q";;
- : cnf = [[(false, "P")]]

# subst [[(false, "P"); (true, "Q")];
             [(true, "R"); (false, "Q"); (true, "S")]] true "P";;
- : cnf = [[(true, "Q")]; [(true, "R"); (false, "Q"); (true, "S")]]
```

(b) BCP(Boolean Constraint Propgation) 함수 `bcp`를 구현하시오.

```
bcp : cnf -> cnf
```

예를 들어,

```
# bcp [[(true, "P")]];;
- : cnf = []

# bcp [[(false, "P")]];;
- : cnf = []

# bcp [
  [(true, "P")];
  [(false, "P"); (true, "Q")];
  [(true, "R"); (false, "Q"); (true, "S")]
];;
- : cnf = [[(true, "R"); (true, "S")]]

# bcp [[(true, "P((P and Q) or R)")];
```

```
 [(false, "P((P and Q) or R)"); (true, "P(P and Q)"); (true, "R")];
 [(false, "P(P and Q)"); (true, "P((P and Q) or R)")];
 [(false, "R"); (true, "P((P and Q) or R)")];
 [(false, "P(P and Q)"); (true, "P")]; [(false, "P(P and Q)"); (true, "Q")];
 [(false, "P"); (false, "Q"); (true, "P(P and Q)")]];;
- : cnf =
[[(true, "P(P and Q)"); (true, "R")]; [(false, "P(P and Q)"); (true, "P")];
 [(false, "P(P and Q)"); (true, "Q")];
 [(false, "P"); (false, "Q"); (true, "P(P and Q)")]]

# bcp [[(true, "P((P -> Q) and (P and (not Q)))")];
 [(false, "P((P -> Q) and (P and (not Q)))"); (true, "P(P -> Q)")];
 [(false, "P((P -> Q) and (P and (not Q)))"); (true, "P(P and (not Q))")];
 [(false, "P(P -> Q)"); (false, "P(P and (not Q))");
  (true, "P((P -> Q) and (P and (not Q)))")];
 [(false, "P(P -> Q)"); (false, "P"); (true, "Q")];
 [(true, "P"); (true, "P(P -> Q)")]; [(false, "Q"); (true, "P(P -> Q)")];
 [(false, "P(P and (not Q))"); (true, "P")];
 [(false, "P(P and (not Q))"); (true, "P(not Q)")];
 [(false, "P"); (false, "P(not Q)"); (true, "P(P and (not Q))")];
 [(false, "P(not Q)"); (false, "Q")]; [(true, "P(not Q)"); (true, "Q")]];;
- : cnf = [[]]
```

(c) PLE(Pure Literal Elimination) 함수 `ple`를 구현하시오.

```
ple : cnf -> cnf
```

예를 들어,

```
# ple [[(true, "P")]];;
- : cnf = []

# ple [[(false, "P")]];;
- : cnf = []

# ple [
  [(true, "P")];
  [(false, "P"); (true, "Q")];
  [(true, "R"); (false, "Q"); (true, "S")]
];;
- : cnf = []

# ple [
  [(true, "P(P and Q)"); (true, "R")];
  [(false, "P(P and Q)"); (true, "P")];
  [(false, "P(P and Q)"); (true, "Q")];
  [(false, "P"); (false, "Q"); (true, "P(P and Q)")]];;
- : cnf =
[[(false, "P(P and Q)"); (true, "P")];
 [(false, "P(P and Q)"); (true, "Q")];
 [(false, "P"); (false, "Q"); (true, "P(P and Q)")]]
```

제출방법 `sat.ml`[1] 파일을 완성한 후 블랙보드에 제출합니다. 모든 코드는 스스로 작성해야 합니다. 본인이 작성하지 않은 코드(e.g., AI, 인터넷, etc)를 제출하는 경우 F 학점이 부여됩니다.

[1] `https://github.com/kupl-courses/COSE419-2024/blob/main/hw1/sat.ml`