Problem Solving using SMT Solver (1)
COSE419, Spring 2024

Hakjoo Oh

Due: 4/19 23:59

Problem 1 (Exact Covering) *다음과 같은 이차원 행렬을 생각하자.*

	1	**2**	**3**	**4**	**5**	**6**	**7**
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

각 가로줄($A, B, \ldots, F$)은 집합 $X = \{1, 2, \ldots, 7\}$의 부분집합을 의미한다. 예를 들어, A는 부분집합 $\{1, 4, 7\}$을 나타낸다. 집합 X의 모든 원소를 포함하되 각 원소가 하나의 부분집합에만 속하게 되는 부분집합들의 집합을 Exact Cover라고 한다. 예를 들어, 위 예제의 경우 $\{B, D, F\}$가 Exact Cover 이며, $\{B, D, E\}$ 또는 $\{A, E\}$ 등은 아니다.

문제를 더 정확하게 정의하면 다음과 같다. 원소들의 집합을 $X = \{x_1, x_2, \ldots, x_m\}$라고 하고 (e.g., $X = \{1, 2, \ldots, 7\}$), $Y = \{y_1, y_2, \ldots, y_n\}$를 주어진 부분집합 이름들의 집합이라고 하자 (e.g., $Y = \{A, B, \ldots, F\}$). 주어진 행렬을 함수 $M : Y \to 2^X$로 나타낼 때(e.g., $M(A) = \{1, 4, 7\}$), 아래 두 조건을 만족하는 Y의 부분집합 $S \subseteq Y$를 X의 Exact Cover라고 한다:

1. *S covers X:*
$$X = \bigcup_{s \in S} M(s) \tag{1}$$
2. *the chosen subsets in $M(s)$ (where $s \in S$) are pairwise disjoint: for all $s_1, s_2 \in S$,*
$$M(s_1) \cap M(s_2) = \emptyset \tag{2}$$

SAT Encoding *부울 변수 X_i $(1 \leq i \leq n)$와 T_{ij} $(1 \leq i \leq n, 1 \leq j \leq m)$를 도입하고 의미를 아래와 같이 정의하자:*

$$X_i \iff s_i \in S, \qquad T_{ij} \iff x_j \in M(s_i)$$

Exact Cover의 두 조건 (1)과 (2)를 각각 논리식 Φ_1과 Φ_2로 표현해 보자:

$$\Phi_1 =$$
$$\Phi_2 =$$

Implementation *입력 타입은 아래와 같다.*

```
type sets = int * int * (int list list)
```

첫번째 정수는 주어진 부분집합들의 개수이다. 두번째 정수는 원소들의 개수이다. 예를 들어, 위 문제는 아래와 같이 정의된다.

```
let sets1 : sets = (6, 7,
[
  [1;0;0;1;0;0;1];
  [1;0;0;1;0;0;0];
  [0;0;0;1;1;0;1];
  [0;0;1;0;1;1;0];
  [0;1;1;0;0;1;1];
  [0;1;0;0;0;0;1];
])
```

문제를 인코딩할 논리식을 아래와 같이 정의하자.

```
type formula =
  | S of int
  | T of int * int
  | Bool of bool
  | And of formula list
  | Or of formula list
  | Not of formula
  | Imply of formula * formula
  | Iff of formula * formula
  | Neq of int * int
```

1. *인코딩 함수* `encode`*를 작성하시오.*

   `encode : sets -> formula`

2. *해를 구하는 함수* `solve`*를 작성하시오.*

   `solve : formula -> int list`

   예를 들어, 위 예제의 경우 `[1; 3; 5]`*가 출력되어야 한다. (부분 집합의 개수를 n이라 할 때, 개별 부분집합들을 각각 정수 $1, 2, \ldots, n$으로 지칭한다.)*

Problem 2 (Learning a Boolean Function) *입출력 명세(input-output specification)로부터 부울 함수를 학습하는 합성기(synthesizer)를 구현해 보자. 다음과 같은 DNF(Disjunctive Normal Form) 형태의 부울 함수를 생각하자:*

$$f(x_1, x_2, \ldots, x_N) = \bigvee_i^M \bigwedge_j l_{ij}$$

여기서 N은 함수의 인자 개수, M은 $\vee$로 연결된 항(term)들의 개수, l_{ij}는 리터럴(변수 x 또는 $\neg x$)을 뜻한다. 예를 들어, 아래의 XOR 함수

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

는 다음과 같이 표현할 수 있다:

$$f(x_1, x_2) = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$$

주어진 입출력 예제를 만족하는 가장 일반적인 부울 함수를 합성하는 프로그램을 작성해 보자.

Examples

1. *위에서 예로 든 XOR 함수의 경우 다음과 같은 명세를 입력 받는다.*

```
2
--positive--
01
10
--negative--
11
00
```

가장 첫 줄의 숫자는 합성할 함수의 인자 개수(N), 두번째 줄의 숫자는 positive/negative 입출력 예제의 개수를 뜻한다. `--positive--` *이후에는 함수의 값을 1로 만드는 입력의 예들 (positive examples)이 주어진다.* `--negative--` *이후에는 함수의 값을 0으로 만드는 입력의 예들(negative examples)이 주어진다. 이러한 입력이 주어졌을때 다음과 같이 모든 입출력 예제들을 만족하는 부울 함수를 출력한다:*

```
f(X) = (!x1 /\ x2) \/ (x1 /\ !x2)
```

2. *AND 함수는 다음과 같이 명세할 수 있다:*

```
2
--positive--
11
--negative--
01
10
00
```

합성기는 다음의 함수를 출력해야 한다:

```
f(X) = (x1 /\ x2)
```

합성 결과는 주어진 예제들을 만족하는 가장 일반적인 함수이어야 한다. "가장 일반적"이란 최소의 항을 가지는 함수를 뜻한다(M이 최소인 함수). 예를 들어, AND 함수는 다음과 같이 나타낼 수도 있지만 가장 일반적인 함수는 아니다:

```
f(X) = (x1 /\ x2) \/ (x1 /\ !x1)
```

3. *아래 명세를 생각하자.*

```
3
--positive--
000
010
100
110
--negative--
001
011
101
111
```

위 명세를 만족하는 가장 일반적인 함수는 다음과 같다:

```
f(X) = (!x3)
```

4. *아래 명세를 생각하자.*

```
4
--positive--
0001
0011
0101
1001
1011
1101
1110
1111
--negative--
0000
0010
0100
0110
0111
1000
1010
1100
```

위 명세를 만족하는 가장 일반적인 함수는 세 개의 항을 가진다:

```
f(X) = (!x3 /\ x4) \/ (x1 /\ x2 /\ x3) \/ (!x2 /\ x4)
```

5. *아래 명세를 생각하자.*

```
20
--positive--
11001001000011111101
10101010001000100001
01101000110000100011
01001100010011000110
01100010100010111000
00001101110000011100
11010001001010010000
00100100111000001000
10001010011001111100
11000111010000000010
00001011101111101010
01100011101100010011
10011011001000100101
00010100101000001000
01111001100011100011
01000000010011011101
--negative--
10101101111110000101
01000101100010100010
10111011010010101001
10101010111111011100
01010110001000000010
```

```
01110011110100111100
11110001110110001011
10011100010110000011
11001110001011010011
01101001010110101001
11100001001101100100
00010001010001100100
00110011111110111100
11001001001110011101
11001110001001001001
10110011111011111001
```

위 명세를 만족하는 가장 일반적인 함수는 세 개의 항을 가진다:

```
f(X) = (!x4 /\ x10 /\ !x12) \/ (!x6 /\ !x10 /\ !x12) \/ (!x1 /\ x9 /\ x11 /\ !x18)
```

SAT Encoding *항의 개수를 M, 인자의 개수를 N이라고 할 때, 부울 변수 p_{ij}와 q_{ij}($1 \leq i \leq M, 1 \leq j \leq N$)를 다음과 같이 정의하자:*

$$p_{ij} \iff \textit{term } i \textit{ contains } x_j, \qquad q_{ij} \iff \textit{term } i \textit{ contains } \neg x_j$$

Positive 예제 개수를 P라고 할 때, 부울 변수 z_{ik} ($1 \leq i \leq M, 1 \leq k \leq P$)를 다음과 같이 정의하자:

$$z_{ik} \iff \textit{term } i \textit{ evaluates to true when the } k^{th} \textit{ positive example is given}$$

주어진 입출력 명세들에 대해서 위 변수들이 성립해야 하는 조건을 기술해 보자. 주어진 명세의 첫 번째 입출력 예제가 $f(1,1,0,0,\ldots,1) = 1$라고 하자. 이 명세는 다음과 같이 변환하면 된다:

1. $z_{11} \vee z_{21} \vee \cdots \vee z_{M1}$
2. $(\neg z_{i1} \vee \neg q_{i1}) \wedge (\neg z_{i1} \vee \neg q_{i2}) \wedge (\neg z_{i1} \vee \neg p_{i3}) \wedge (\neg z_{i1} \vee \neg p_{i4}) \wedge \cdots \wedge (\neg z_{i1} \vee \neg q_{iN})$ *for* $1 \leq i \leq M$

입출력 예제가 $f(1,0,1,0,\ldots,1) = 0$을 포함한다고 해 보자. 이 명세는 다음과 같이 표현할 수 있다:

$$\bigwedge_{i=1}^{M} (q_{i1} \vee p_{i2} \vee q_{i3} \vee p_{i4} \vee \cdots \vee q_{iN})$$

Implementation *명세를 다음과 같이 OCaml 타입으로 정의하자:*

```
type specification = int * int * example list * example list
and example = int list
```

첫번째 정수는 합성할 함수의 최대 항 개수이다 (M의 최대치). 이 숫자 이하의 항을 가지는 함수가 없다면 합성에 실패하도록 한다. 예를 들어, XOR 함수는 아래와 같이 명세할 수 있다:

```
let spec1 = (10, 2,
[
  [0; 1];
  [1; 0]
],
[
  [1; 1];
  [0; 0]
])
```

DNF는 아래의 데이터타입으로 정의한다:

```
type dnf = conj list
and conj = lit list
and lit = X of int | NotX of int
```

명세로부터 DNF를 합성하는 함수 `synthesize`*를 작성하시오.*

`synthesize : specification -> dnf option`

예를 들어, 위의 XOR명세가 주어질 경우 `Some [[NotX 1; X 2]; [X 1; NotX 2]]`*을 리턴한다. 명세를 만족하는 함수가 없다면* `None`*을 리턴한다.*