Ben-Gurion University of the Negev
Department of Computer Science
Logic programming

Assignment 2, Semester B, 2023

Deadline: May 23 at 23:00

SAT Solving with Binary Numbers

In this exercise you are required to encode operations on binary numbers to CNF, and apply a SAT solver in order to solve several problems.

We represent the truth values `true` and `false` as the values `1` and `-1`, respectively. A bit vector is a (non-empty) list consisting of Boolean variables and/or truth values. A (known or unknown) binary number is represented as a bit vector in which the first element represents the least significant bit (LSBF). For example: the bit vectors [1,1,-1,1] and [1,1,-1,1,-1,-1] both represent the number 11. The bit vector [X,Y,Z] represents an unknown binary number with 3 bits. The bit vector [1,Y,Z] represents an odd binary number with 3 bits, and the bit vector [-1,Y,Z] represents an even binary number with three bits.

In this assignment, all operations on binary numbers operate on fixed length bit vectors. If the operation has an output, then the output is of the same length as the inputs to the operation. Overflow is considered an error (failure).

Task 1: Binary to Decimal and back again

You are to write a Prolog predicate `binary2decimal(Xs,N)` with mode `binary2decimal(+,-)` which succeeds if and only if `N` is the decimal representation of the binary number `Xs`. For example,

```
?- binary2decimal([-1,1,-1],N).
N = 2.

?- binary2decimal([-1,-1,1,-1],N).
N = 4.
```

You are to write a Prolog predicate `decimal2binary(N,Xs)` with mode `decimal2binary(N,Xs)(+,-)` which succeeds if and only if `Xs` is the binary representation (with no leading zeroes) of the decimal number `N`. For example,

```
?- decimal2binary(4,Xs).
Xs = [-1, -1, 1] .

?- decimal2binary(0,Xs).
Xs = [-1] .
```

Task 2: Encoding Addition

You are to write a Prolog predicate `add(Xs,Ys,Zs,Cnf)` which encodes binary addition. The predicate expects Xs and Ys to be bound to length k bit vectors. It creates the length k bit vector Zs and a Cnf which is satisfied precisely when Xs,Ys,Zs are bound to binary numbers such that the sum of Xs and Ys is Zs. For example:

```
?- Xs=[_,_], Ys=[_,_], add(Xs,Ys,Zs,  Cnf), sat(Cnf).
Xs = [1, -1],
Ys = [1, -1],
Zs = [-1, 1],
Cnf = ...

?- Xs=[1,_,_], Ys=[_,_,_], add(Xs,Ys,Zs,  Cnf), sat(Cnf).
Xs = [1, 1, -1],
Ys = [-1, 1, -1],
Zs = [1, -1, 1],
Cnf = ...

?- Xs=[1,_,_,_], Ys=[-1,1,1,1], add(Xs,Ys,Zs,  Cnf), sat(Cnf).
Xs = [1, -1, -1, -1],
Ys = [-1, 1, 1, 1],
Zs = [1, 1, 1, 1],
Cnf = ...

?- Xs=[_,1,_,_], Ys=[-1,1,1,1], add(Xs,Ys,Zs,  Cnf), sat(Cnf).
false.
```

Task 3: Encoding less equals, less than, and sorted

You are to write the Prolog predicates `leq(Xs,Ys,Cnf)` and `lt(Xs,Ys,Cnf)` which encode the binary relations less equal and less than. The predicates expect Xs and Ys to be bound to length k bit vectors. They create Cnfs which are satisfied precisely when Xs is less equal Ys and when Xs is less than Ys. For example:

```
?- Xs=[1,_], Ys=[_,_], leq(Xs,Ys,Cnf), sat(Cnf).
Xs = Ys, Ys = [1, -1],
Cnf = ...

?- Xs=[1,_], Ys=[_,_], lt(Xs,Ys,Cnf), sat(Cnf).
Xs = [1, -1],
Ys = [-1, 1],
Cnf =  ...

?- Xs=[_,1], Ys=[_,-1], lt(Xs,Ys,Cnf), sat(Cnf).
false.
```

You are to write the Prolog predicate `sorted(Vectors,Cnf)` which encodes that `Vectors` represents a sorted list of binary numbers. The predicate expects `Vectors` to be a list of fixed length bit vectors. It creates a Cnf that is satisfied precisely when `Vectors` represents a sorted list of strictly increasing binary numbers. For example:

```
?- Vectors=[[_,_],[_,_],[_,_]], sorted(Vectors,Cnf), sat(Cnf).
Vectors = [[-1, -1], [-1, 1], [1, 1]],
Cnf = ...
```

Task 4: Encoding different and allDifferent

You are to write the Prolog predicate `diff(Xs, Ys, Cnf)` which encodes that Xs and Ys are different (not equal). The predicate expects Xs and Ys to be fixed length bit vectors. It creates a Cnf that is satisfied precisely when Xs and Ys represent different binary numbers. For example:

```
?- Xs=[_,_], Ys=[_,_], diff(Xs,Ys,Cnf), sat(Cnf).
Xs = [1, 1],
Ys = [-1, -1],
Cnf = ...
```

You are to write the Prolog predicate `allDiff(Vectors,Cnf)` which encodes that `Vectors` represents a list of different binary numbers. The predicate expects `Vectors` to be a list of fixed length bit vectors. It creates a Cnf that is satisfied precisely when `Vectors` represents a list of all different binary numbers. For example:

```
?- Vectors=[[_,_],[_,_],[_,_],[_,_]], allDiff(Vectors,Cnf), sat(Cnf).
Vectors = [[-1, 1], [1, -1], [-1, -1], [1, 1]],
Cnf = ...
```

Sat Solving for Golomb Rulers

Golomb Ruler: A Golomb ruler is a list `Xs` of increasing integers such that no two pairs of elements are the same distance apart. The length of `Xs` is called the order of the ruler. The smallest element is customarily put at 0.

In this part of the assignment we will apply a sat solver to construct Golomb rulers. A problem instance takes the form `golomb(N,Max)` and a solution is a Golomb ruler of length `N` in which all elements are less or equal to `Max`.

The structure of your code should be:

```
solve(Instance, Solution) :-
    encode(Instance,Map,Cnf),
    sat(Cnf),
    decode(Map,Solution),
    verify(Instance, Solution).
```

For example:

```
?- solve(golomb(4,6),Solution).
verify:ok
Solution = [0, 2, 5, 6] .

?- solve(golomb(4,5),Solution).
false.
```

Task 5: verify

You are to write the Prolog predicate `verify(Instance,Solution)` with mode `verify(+,+)`. Given `Instance = golomb(N,Max)` and a list of numbers `Solution`, the call to `verify(Instance,Solution)` writes one line:

- "verify:ok", if Solution is a Golomb ruler of length `N` in which all elements are less or equal to `Max`; and
- "verify:wrong", otherwise.

```
?- verify(golomb(4,6),[0,2,5,6]).
verify:ok
true.

?- verify(golomb(4,6),[0,2,5]).
verify:wrong
true.

?- verify(golomb(4,6),[0,2,4,6]).
verify:wrong
true.

?- verify(golomb(4,6),[0,5,2,6]).
verify:wrong
true.
```

Task 6: encode

You are to write the Prolog predicate `encode(Instance,Map,Cnf)`. Given an instance, `Instance = golomb(N,Max)` a call to this predicate generates a structure Map and formula Cnf such that Cnf is satisfied exactly when the instance has a solution and it can be decoded from Map. One possible choice for the structure of Map is as in the following example:

```
?- encode(golomb(4,6),Map,Cnf), sat(Cnf).
Map = [[-1, -1, -1], [-1, 1, -1], [1, -1, 1], [-1, 1, 1]],
Cnf = ...
```

Putting it all together

The following are some examples (running on a solution to this assignment). We use the following predicate for timing (we will not run your program for more than 30 sec):

```
time(X) :-
    statistics(cputime,Time1),
    (call(X) -> writeln(true) ; writeln(false)),
    statistics(cputime,Time2),
    Time12 is Time2-Time1,
    writeln(Time12:sec).

?- time(solve(golomb(7,25),Solution)).
verify:ok
true
0.046632403999865346:sec
Solution = [0, 2, 6, 9, 14, 24, 25].

?- time(solve(golomb(7,24),Solution)).
false
0.33383753899988733:sec
true.

?- time(solve(golomb(9,44),Solution)).
verify:ok
true
11.086411146000046:sec
Solution = [0, 3, 9, 17, 19, 32, 39, 43, 44].

?- time(solve(golomb(9,43),Solution)).
false
37.4876510400004:sec
true.

?- time(solve(golomb(10,55),Solution)), writeln(Solution).
verify:ok
true
90.00485191799999:sec
Solution = [0, 2, 14, 21, 29, 32, 45, 49, 54, 55].
```

Grading & Procedures

Evaluation :

This assignment has 6 tasks. We will grade only 5 of them and give 20 points for each of these 5 tasks. Note that many tasks rely on code from other tasks. So you

will need to solve all tasks.

After Solving :

When grading your work, an emphasis will be given on code efficiency and readability. We appreciate effective code writing. The easier it is to read your code — the more we appreciate it! Even if you submit a partial answer. So please indent your code, add good comments.

Procedure

Submit a single file called `ex2.pl` with the assignment's solution. Please include a header with following statement:

> /**** I, Name (ID number) assert that the work I submitted is entirely my own. I have not received any part from any other student in the class (or other source), nor did I give parts of it for use to others. I have clearly marked in the comments of my program any code taken from an external source. *****/

Submission is solo, i.e., you may *not* work in pairs. If you take any parts of your solution from an external source you must acknowledge this source in the comments. Please note that we test your work using a Linux installed SWI-Prolog (as in the CS Labs) – so please make sure your assignment runs on such a configuration.

Your documentation should detail the limits of your solution.