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

Assignment 2, Semester B, 2022

Deadline: May 31 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.

Task 1: Encoding Addition (10%)

You are to write a Prolog predicate `add(Xs,Ys,Xs,Cnf)` which encodes binary addition. The predicate expects Xs and Ys to be bound to bit vectors. It creates the bit vector Zs (with a sufficient number of bits) 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=[1,_], Ys=[_,_,_], add(Xs,Ys,Zs,Cnf), sat(Cnf).
Xs = [1, -1],
Ys = [1, -1, -1],
Zs = [-1, 1, -1, -1],
 ...
```

If we want to apply the predicate `add(Xs,Ys,Zs,Cnf)` in the "backwards" direction: giving the bit vector Zs, we need to bind Xs and Ys to bit vectors and take into consider the possible lengths of the bit vectors. In the following example, Zs is given in 4 bits, hence Xs and Ys might each have 4 bits as well. However, the encoding of addition expects the sum to have one additional bit. So we pad Zs with one aditional (false) bit:

```
?- Zs=[1,-1,1,1], PaddedZs=[1,-1,1,1,-1], length(Xs,4), length(Ys,4),
   add(Xs,Ys,PaddedZs,Cnf), sat(Cnf).
Xs = [-1, 1, 1, -1],
Ys = [1, 1, 1, -1],
 ...
```

Task 2: Encoding less equals and less than (10%)

You are to write a Prolog predicate `leq(Xs,Ys,Xs,Cnf)` which encodes the binary relation less equal. The predicate expect Xs and Ys to be bound to bit vectors. It creates a Cnf which is satisfied precisely when Xs is less equal Ys. For example:

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

You are to write a Prolog predicates `lt(Xs,Ys,Xs,Cnf)` which encodes the binary relationsless than. The predicates expect Xs and Ys to be bound to bit vectors. It creates a Cnf which is satisfied precisely when Xs is less than Ys. For example:

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

Task 3: Encoding Sum (10%)

You are to write a Prolog predicate `sum(ListofNumbers, Zs, Cnf)` which encodes the sum of a list of binary numbers. The predicate expects List to be bound to a list of bit vectors. It creates the bit vector Zs and a Cnf which is satisfied when The vectors in List and Zs are bound to binary numbers such that the sum of the numbers in List is equal to Zs. For example:

```
?- length(Xs1,5), length(Xs2,5), length(Xs3,5), sum([Xs1,Xs2,Xs3],Zs,Cnf),
   sat([Xs1,Xs2,Xs3|Cnf]).
Xs1 = [-1, -1, -1, 1, -1],
Xs2 = [-1, 1, -1, -1, -1],
Xs3 = [1, -1, -1, -1, -1],
Zs = [1, 1, -1, 1, -1, -1, -1, -1, -1],
 ...
```

If we want to apply the predicate `sum(List,Zs,Cnf)` in the “backwards” direction: giving the bit vector Zs, we need to bind List to a list of bit vectors and take into consider the possible different lengths of the bit vectors. For example:

```
?- Zs = [-1,1,1,1,1], PaddedZs= [-1,1,1,1,1,-1,-1,-1,-1], List = [Xs1, Xs2, Xs3],
   length(Xs1,5), length(Xs2,5), length(Xs3,5),
   sum(List,PaddedZs,Cnf), sat(Cnf).
List = [[-1, -1, -1, -1, -1], [1, 1, 1, 1, -1], [1, 1, 1, 1, -1]],
Xs1 = [-1, -1, -1, -1, -1],
Xs2 = [1, 1, 1, 1, -1],
Xs3 = [1, 1, 1, 1, -1],
 ...
```

Task 4: Encoding Multiplication (10%)

You are to write a Prolog predicate `times(Xs,Ys,Xs,Cnf)` which encodes binary multiplication. The predicate expects Xs and Ys to be bound to bit vectors. It creates the bit vector Zs and a Cnf which is satisfied when Xs,Ys,Zs are bound to binary numbers such that the multiplication of Xs and Ys is Zs. In the following example when we call the sat solve as in sat([Xs,Ys—Cnf]), we are constraining Xs and Ys to represent positive numbers. For example:

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

If we want to apply the predicate `times(Xs,Ys,Zs,Cnf)` in the “backwards” direction: giving the bit vector Zs, we need to bind Xs and Ys to bit vectors and take into consider the possible lengths of the bit vectors. For example:

```
?- Zs=[1,1,1,1], PaddedZs=[1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
   length(Xs,4), length(Ys,4), times(Xs,Ys,PaddedZs,Cnf), sat(Cnf).
Xs = [1, 1, -1, -1],
Ys = [1, -1, 1, -1],
 ...
```

Task 5: Encoding Power (10%)

You are to write a Prolog predicate `power(N,Xs,Zs,Cnf)` which encodes binary exponent. The predicate expects Xs and Ys to be bound to bit vectors. It creates the bit vector Zs and a Cnf which is satisfied when Xs,Zs are bound to binary numbers such that the Nth power of Xs is Zs. In the example, when we write sat([Xs—Cnf]) we are constraining Xs to be positive. For example:

```
?- Xs=[_,_,_], power(3,Xs,Zs,Cnf), sat([Xs|Cnf]).
Xs = [-1, 1, -1],
Zs = [-1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
 ...
```

Note: In this task part of your grade will relate to the size (number of clauses) of the CNF encoding. For the previous example, there is a solution which creates a CNF with less than 500 clauses. For the similar query with `power(7,Xs,Zs,Cnf)`, there is an encoding with less than 9000 clauses.

Task 6: Encoding Power Equations(10%)

An (n, m) power equation is an equation of the form

$$b^n = a_1^n + \cdots a_m^n$$

You are to write a Prolog predicate `powerEquation(N,M,Zs,List,Cnf)` which given positive numbers N and M and a bit vector Zs generates a list $List = [As1, \ldots, As_M]$ and a Cnf which is satisfied exactly when $Zs^N = As1^N + \cdots + As_M^N$. The power equation found in the following example is $5^2 = 3^2 + 4^2$.

Example

```
?- Zs=[_,_,_], powerEquation(2,2,Zs,List,Cnf), sat([Zs|Cnf]).
Zs = [1, -1, 1],
List = [[1, 1, -1], [-1, -1, 1]],
 ...
```

Task 7: Euler's Conjecture(20%)

About 200 years ago Euler conjectured that for that for all integers n and k greater than 1, if the sum of n many k^{th} powers of positive integers is itself a kth power, then n is greater than or equal to k. Here is the conjecture in symbols:

$$b^n = a_1^n + \cdots a_n^n \Rightarrow n \geq k$$

In 1966, this conjecture was disproved for $k = 5$ and in 1988 it was disproved for $k = 4$. It is unknown whether the conjecture fails or holds for any value $k \geq 6$. Two of the known counter examples found are:

1. $144^5 = 27^5 + 84^5 + 110^5 + 133^5$
2. $422481^4 = 95800^4 + 217519^4 + 414560^4$

You are to write a predicate `solve(Instance,Solution)`. Given $Instance = euler(N, NumBits)$ where `N` is the given power and `NumBits` is a number of bits, a solution is a list of positive numbers of the form $[B, A_1, \ldots, A_{N-1}]$ such that $B^N = A_1^N + \cdots A_{N-1}^N$, $A_1 \leq A_2 \leq \cdots \leq A_{N-1}$, and such that all of the numbers can be represented as bit vectors in `NumBits`. Each such solution is a counter example to the above conjecture. If there is no solution for a given instance, then the call to `solve(Instance,Solution)` should fail. In all of the given examples, the times are given in seconds.

The structure of your code should be:

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

?- statistics(cputime,Time1),   solve(euler(5,8), Solution),
   statistics(cputime,Time2),   Time12 is floor(Time2-Time1).
verify:ok
Solution = [144, 27, 84, 110, 133],
Time12 = 57
 ...

?- solve(euler(5,7), Solution).
false.
```

Task 8: Power Partition — All Solutions(20%)

The n^{th} power partition of a number b is a sequence of n numbers $(a_1, a_s, \ldots, a_n)$ such that

$$b^n = a_1^n + \cdots a_n^n$$

There are several known solutions to such equations (see examples below). There are no solutions for $n = 6$.

You are to write a predicate `solveAll(Instance,Solution)`. Given $Instance = partition(N, NumBits)$ where N and NumBits are positive numbers. A solution is a list of lists each of which consists of $N + 1$ positive numbers of the form $[B, A_1, \ldots, A_{N-}]$ such that $B^N = A_1^N + \cdots A_N^N$, $A_1 \leq A_2 \leq \cdots \leq A_{N-1}$, and such that all of the numbers in the list can be represented as bit vectors in NumBits. Each solution in the list describes a number B with its n^{th} power partition. If there are no solutions for an instance partition(N,NumBits) then the call should bind $Solution = [\,]$.

The structure of your code should be:

```
solveAll(Instance,Solutions) :-
    encode(Instance,Map,Cnf),
    satMulti(Cnf,1000,_Count,_Time),
    decodeAll(Map,Solutions),
    verifyAll(Instance,Solutions).
```

The following are some examples (running on a solution to this assignment):

N=2

```
?- statistics(cputime,Time1), solveAll(partition(2,5), Solutions),
   statistics(cputime,Time2),   Time12 is Time2-Time1.
```

```
verify:ok
Solutions = [[5,3,4],[10,6,8],[13,5,12],[15,9,12],[17,8,15],[20,12,16],
             [25,7,24],[25,15,20],[26,10,24],[29,20,21],[30,18,24]].
Time12 = 0.018943040999999994 .
```

N=3

```
?- statistics(cputime,Time1),   solveAll(partition(3,4), Solutions),
    statistics(cputime,Time2),   Time12 is (Time2-Time1).
verify:ok
Solutions = [[6, 3, 4, 5], [9, 1, 6, 8], [12, 6, 8, 10]],
Time12 = 0.07964820699999997
```

N=4

```
?- statistics(cputime,Time1),   solveAll(partition(4,9), Solutions),
   statistics(cputime,Time2),   Time12 is (Time2-Time1).
verify:ok
Solutions = [[353, 30, 120, 272, 315]],
Time12 = 7795.691558663
```

N=5

```
?- statistics(cputime,Time1),   solveAll(partition(5,7), Solutions), writeln(Solutions),
verify:ok
Solutions = [[107, 7, 43, 57, 80, 100], [94, 21, 23, 37, 79, 84],
             [72, 19, 43, 46, 47, 67]],
Time12 = 1686 .
```

1 Grading & Procedures

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. BBBBB