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

Assignment 3, Semester B, 2023

Deadline: June 18 at 23:00

Representing Numbers in the Order Encoding

In this exercise you are required to encode operations on unary numbers in the order encoding 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) unary number in the order encoding is represented as a bit vector in which all of the values "1" come before all of the values "-1". The number represented by the bit vector is the number of "1" values that occur. For example: the bit vectors [1,1,-1,-1] and [1,1,-1,-1,-1,-1] both represent the number 2. The bit vector [X,Y,Z] might represent an unknown unary number with 3 bits on condition that the values of the variables X,Y and Z are constrained so that all of the values "1" come before all of the values "-1". The bit vectors [-1,-1,-1] and [-1] both represent the number zero.

Task 1: representing numbers

You are to write the Prolog predicate number(LB,UB,Vector,Cnf) which given the values LB (lower bound) and UB (upper bound) such that $LB \leq UB$ creates a bit vector Vector and a formula Cnf such that Cnf is satisfied exactly when Vector represents a number in the order encoding. For example,

```
?- number(1,9,Vector,Cnf), sat(Cnf).
Vector = [1, -1, -1, -1, -1, -1, -1, -1, -1],
Cnf = ...
```

Task 2: adding numbers

You are to write the Prolog predicate add(Xs,Ys,Zs,Cnf) which given bit vectors Xs and Ys which represent numbers in the order encoding, generate a bit vector Zs and a formula Cnf such that Cnf is satisfied exactly when Zs represents a number in the order encoding which is the sum of the numbers represented by Xs and Ys. For example,

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

```
?- number(2,4,Xs,Cnf1), number(1,3,Ys,Cnf2), add(Xs,Ys,Zs,Cnf3),
   append([Cnf1,Cnf2,Cnf3],Cnf), sat(Cnf).
Xs = [1, 1, 1, -1],
Ys = [1, -1, -1],
Zs = [1, 1, 1, 1, -1, -1, -1],
Cnf = ...
```

Task 3: Encoding different and allDifferent

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

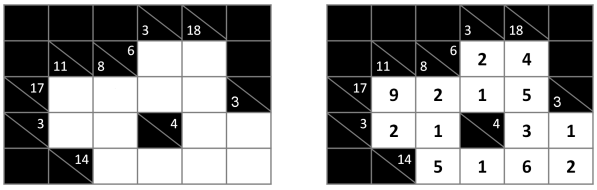
```
?- number(0,3,Xs,Cnf1), number(1,2,Ys,Cnf2),
   diff(Xs,Ys,Cnf3), append([Cnf1,Cnf2,Cnf3],Cnf), sat(Cnf).
Xs = [1, -1, -1],
Ys = [1, 1],
Cnf = ...
```

You are to write the Prolog predicate `allDiff(Vectors,Cnf)` which given a list `Vectors` of bit vectors representing numbers in the order encoding, creates a Cnf that is satisfied precisely when `Vectors` represents a list of all different numbers. For example:

```
?- number(0,2,Xs1,Cnf1), number(0,1,Xs2,Cnf2), number(2,2,Xs3,Cnf3),
   allDiffs([Xs1,Xs2,Xs3],Cnf4),append([Cnf1,Cnf2,Cnf3,Cnf4],Cnf),sat(Cnf).
Xs1 = [-1, -1],
Xs2 = [1],
Xs3 = [1, 1],
Cnf = ...
```

Kakuro

In the game of Kakuro you are required to fill horizontal and vertical blocks with distinct integers between 1 and 9 which sum to a given clue. (you can read more about the game of Kakuro on Wikipedia) and here is an example Kakuro board (left) with its solution (right):



We represent Kakuro instances as a list of blocks. Each element of the list is of the form `ClueSum = Block`, where `ClueSum` is an integer and `Block` is a list of the block's variables.

Below is our representation for the Kakuro board in the previous example. Note that some variables occur twice, once in a "across block" and once in a "down block".

$$\left[\begin{array}{l} 6 = [\mathtt{I}_1, \mathtt{I}_2],\ 17 = [\mathtt{I}_3, \mathtt{I}_4, \mathtt{I}_5, \mathtt{I}_6],\ 3 = [\mathtt{I}_7, \mathtt{I}_8],\ 4 = [\mathtt{I}_9, \mathtt{I}_{10}],\ 14 = [\mathtt{I}_{11}, \mathtt{I}_{12}, \mathtt{I}_{13}, \mathtt{I}_{14}], \\ 11 = [\mathtt{I}_3, \mathtt{I}_7],\ 8 = [\mathtt{I}_4, \mathtt{I}_8, \mathtt{I}_{11}],\ 3 = [\mathtt{I}_1, \mathtt{I}_5],\ 18 = [\mathtt{I}_2, \mathtt{I}_6, \mathtt{I}_9, \mathtt{I}_{13}],\ 3 = [\mathtt{I}_{10}, \mathtt{I}_{14}]] \end{array}\right]$$

A Kakuro solution is a list of terms as described above, which contains distinct integers between 1 and 9 instead of variables, such that each block's sum is equal to the clue. Below is our representation for the solution of the Kakuro board in the previous example:

$$\left[\begin{array}{l} 6 = [2, 4],\ 17 = [9, 2, 1, 5],\ 3 = [2, 1],\ 4 = [3, 1],\ 14 = [5, 1, 6, 2], \\ 11 = [9, 2],\ 8 = [2, 1, 5],\ 3 = [2, 1],\ 18 = [4, 5, 3, 6],\ 3 = [1, 2]] \end{array}\right]$$

When solving Kakuro instances we will use the following format:

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

For example:

```
?- Instance=[6=[A,B],17=[C,D,E,F],3=[G,H],4=[I,J],14=[K,L,M,N],
             11=[C,G],8=[D,H,K],3=[A,E],18=[B,F,I,M],3=[J,N]],
|  solve(Instance,Solution).
verify:ok
Solution = [6=[2,4],17=[9,2,1,5],3=[2,1],4=[3,1],14=[5,1,6,2],
            11=[9,2],8=[2,1,5],3=[2,1],18=[4,5,3,6],3=[1,2]]
```

Task 4: verify

You are to write the Prolog predicate `verify(Instance,Solution)` with mode `verify(+,+)`. Given a Kakuro Instance and a Solution, the call to `verify(Instance,Solution)` writes one line:

- "verify:ok", if Solution is a solution to the given Kakuro Instance; and
- "verify:wrong", otherwise.

When solving this task you will notice that the call to the SAT solver assigns values to the variables in the instance.

Task 5: encode

Write a Prolog predicate `encode(Instance,Map,Cnf)` with mode `encode(+,-,-)`. The predicate takes an instance of Kakuro, and encodes it to a `Cnf` which is satisfiable if and only if the mapping of the variables specified in `Map` is a solution to the Kakuro instance.

Task 6: Putting it all together

We will test your submission using the predicate `solve(Instance, Solution)` as specified above. In this task you should all predicates required to make that work.

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 `ex3.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.