Intelligent System - HW2

Vinh T. Nguyen. Email: vinh.nguyen@ttu.edu

September 27, 2018

1 Question 1

Formulate the following problem as a CSP problem. Follow the format of the CSP model of Sudoku problem in the slides.

Each character in the diagram above represents a single digit and different characters represent different values. For example, SEND represents a number with 4 digits. The most significant digits are not equal to zero. Find the value of each character. You may use "=", "!=", and arithmetic expression in your modeling.

Answer:

Variables: = { S, E, N, D, M, O, R, Y }

Domain: = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }. % All variables have the same domain

Constraints:

- S "!=" E "!=" N "!=" D "!=" M "!=" O "!=" R "!=" Y. % Different characters represent different values. Or we can state in a more general way. For any two variables V1, V2, V1 "!=" V2.
- S "!=" 0 and M "!=" 0. % The most significant digits are not equal to zero.
- 1000*S + 100*E + 10*N + D + 1000*M + 100*O + 10*R + E = 10000*M+ 1000*O+ 100*N + 10*E + Y. % Arithmetic expression of adding two strings.
- D + E = 10*Carry1 + Y. %Right most operation with carry
- N + R = Carry1 + 10*Carry2 + E. %Operation with carry
- E + O = Carry2 + 10*Carry3 + N. %Operation with carry
- S + M = Carry3 + 10*Carry4 + O. %Operation with carry
- M = Carry4.%Left most operation
- Carry4 = 1. %Carry4 can only be 1.

2 Question 2

Write ASP model to represent the CSP model of Sudoku program in the slides. You ASP model should have three sections: objects, relations and knowledge/rules. You have to use sameBlock(x1, y1, x2, y2) which denotes that the square at (x1, y1) and the square at (x2, y2) are in the same 3 by 3

block. When you write your knowledge and rules, you may follow the class discussion and practice the top down and iterative refinement methodologies. Refer to the slides for proper solution format and ideas.

Answer:

Objects:

- index: = 0..8. Indexing cell position
- number: = 1..9. % Domain of cell's values
- div: = 0..4. % long division by 3

Relations:.

%square(X,Y,N) denotes that the value of square at row X, column Y is N.

square(#index, #index, #number).

%div(X,N) denotes that the number of divides in long division X by 3 is N. Example, $\{0,1,2\}$ div 3 = 0, $\{3,4,5\}$ div 3 = 1, $\{6,7,8\}$ div 3 = 2.

div(#index, #div).

Knowledge/rules:

% Initially, all squares have all possible values from 1 to 9.

$square(X,Y,1)|square(X,Y,2)|square(X,Y,3)|square(X,Y,4)|$
$square(X,Y,5)|square(X,Y,6)|square(X,Y,7)|square(X,Y,8)|square(X,Y,9).$

% For all X, N is the long division of X by 3.

$div(X,N) : -X/3 = N.$

% For all pairs X,Y. (X,Y) is in the same block if they have the same long division by 3.

$sameBlock(X1,Y1,X2,Y2) : -div(X1,N1), div(X2,N1), div(Y1,N1), div(Y2,N1)$

% Numbers of the same row X can not have the same value N.

$: -square(X,Y1,N), square(X,Y2,N), Y1! = Y2.$

% Numbers of the same column Y can not have the same value N.

$: -square(X1,Y,N), square(X2,Y,N), X1! = X2.$

% Numbers of the same block can not have the same value N.

$: -square(X1,Y1,N), square(X2,Y2,N), sameBlock(X1,Y1,X2,Y2), X1! = X2, Y1! = Y2.$

3 Question 3

Algorithm and its complexity. Give a rigorous proof of the complexity of the arc consistency algorithm on the slides. Your statements should be mathematically rigorous.

In short, the number of constraints is e, since constraints or arcs between variables that are related by symmetric constraints, so we have a total of $2e$ arcs. For each ARC, the algorithm has to CHECK for every value a in the domain of x (we have at most d values) against every value b in the domain of y (we have at most d values) for consistency. The number of at most CHECKS is $d * d$ or d^2. For each CHECK a value a in domain x, if a value a is removed from domain x, the algorithm adds all the arcs of constraints pointing to a. Since we have at most d values in domain

x, we can remove at most d values from domain of x. In total, we have $2e * d^2 * d$ computation. Hence, the complexity of the arc-3 algorithm is $O(ed^3)$

4 Question 4

Given a CSP with 5 variables, each variable has a domain with 3 values.

- 1) What is the size of the search tree for this problem? (Refer to slides for what is a search tree for a CSP). *At level 1, first variable (V1) has 3 choices, for each choice of V1, we have 3 choices of V2, for each choice of V2 we have 3 choice of V3, for each choice of V3 we have 3 choice of V4, for each choice of V4 we have 3 choice of V5. In total we have 3*3*3*3*3 =* 3^5 *combinations. Hence the size of the search tree is* 3^5
- 2) If the CSP above has n variables and the maximum domain size is d, what is the size of the search tree of this problem? *From the previous problem, we can generalize the problem of the size of search tree to* d^n

5 Question 5

The algorithm backtracking search + forward checking can be traced using a tree. As an example, see the following picture. The CSP problem is inside the rectangle, and the tracing tree is at the right side. Note that the current domain of unassigned variables are given in each node.

Answer: The answer is shown in Fig. 1

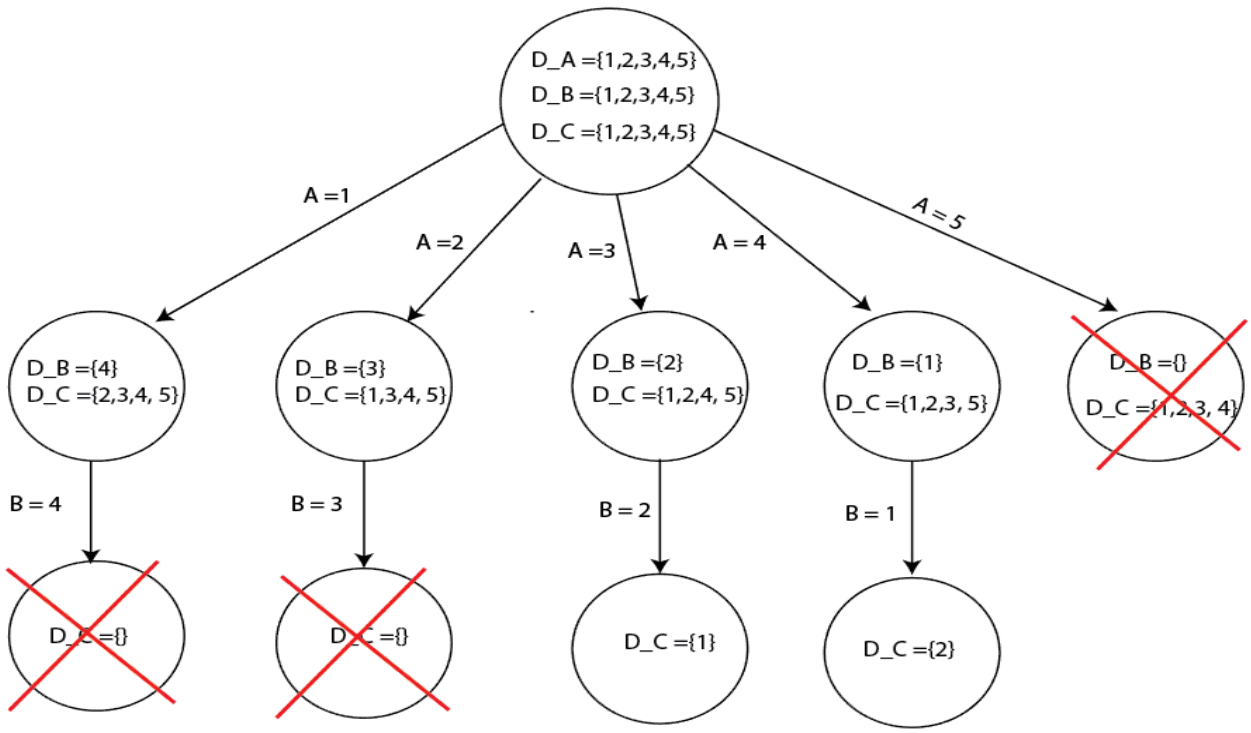

Figure 1: Trace the backtracking + forward checking algorithm

6 Question 6

Extend the backtracking algorithm in the slides with forward checking ability

```
BT (V, D, C) {
    i ← 1; Di' ← Di;            %start at variable x1 and domain D1%
    while (1 <= i <=n) {        %keep doing until all variables are assigned. %
        If Di' is not empty {
            select and delete a value from Di' and assign it for xi.
%Prune the domain of its unassigned neighboring variables. We start from i and increment one by each step, the unassigned neighboring variables thus start from i <k<=n.%
            for all i < k <=n {          % scanning all unassigned variables. %
                for all values b in Dk' {
                    if x1x2 xi b violate any constraints {
                        remove b from Dk'
                    }
                }
                if Dk' is empty {            % perform backtracking %
                    restore Dk' to its prev domain values before xi is assigned.
                    i ← i-1
                }
            }
            i ← i + 1
            Di' ← Di
        }
    }
    if (i==0) report no solution
    else return x1, x2, …, xn as a solution
}
```

Figure 2: Backtracking + Forward checking algorithm