International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:7, No:8, 2013

Modelling Sudoku Puzzles as Block-world Problems

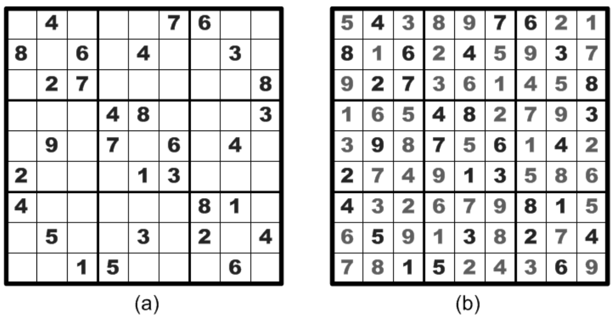
Cecilia Nugraheni and Luciana Abednego

Abstract—Sudoku is a kind of logic puzzles. Each puzzle consists of a board, which is a 9×9 cells, divided into nine 3×3 subblocks and a set of numbers from 1 to 9. The aim of this puzzle is to fill in every cell of the board with a number from 1 to 9 such that in every row, every column, and every subblock contains each number exactly one. Sudoku puzzles belong to combinatorial problem (NP complete). Sudoku puzzles can be solved by using a variety of techniques/algorithms such as genetic algorithms, heuristics, integer programming, and so on. In this paper, we propose a new approach for solving Sudoku which is by modelling them as *block-world problems*. In block-world problems, there are a number of boxes on the table with a particular order or arrangement. The objective of this problem is to change this arrangement into the targeted arrangement with the help of two types of robots. In this paper, we present three models for Sudoku. We modellized Sudoku as *parameterized multi-agent systems*. A parameterized multi-agent system is a multi-agent system which consists of several uniform/similar agents and the number of the agents in the system is stated as the parameter of this system. We use Temporal Logic of Actions (TLA) for formalizing our models.

Keywords—Sudoku puzzle, block world problem, parameterized multi agent systems modelling, Temporal Logic of Actions.

I. Introduction

Sudoku is a kind of logic puzzles. Each puzzle consists of a board, which is a 9×9 cells, divided into nine 3×3 subblocks and a set of numbers from 1 to 9. The aim of this puzzle is to fill in every cell of the board with a number from 1 to 9 such that in every row, every column, and every subblock contains each number exactly one. Sudoku puzzles first popularized a Japanese puzzle company named Nikoli in 1986 and then became known worldwide in 2005. Until now, this puzzle is still very popular. It can be found, for example, in newspapers, like the one given in Figure 1 which taken from one Indonesian newspaper *Kompas*.



(a)

	4				7	6		
8		6		4			3	
	2	7						8
			4	8				3
	9		7		6		4	
2				1	3			
4						8	1	
	5			3		2		4
		1	5				6	

(b)

5	4	3	8	9	7	6	2	1
8	1	6	2	4	5	9	3	7
9	2	7	3	6	1	4	5	8
1	6	5	4	8	2	7	9	3
3	9	8	7	5	6	1	4	2
2	7	4	9	1	3	5	8	6
4	3	2	6	7	9	8	1	5
6	5	9	1	3	8	2	7	4
7	8	1	5	2	4	3	6	9

Fig. 1. A Sudoku puzzle (a) and its solution (b).

Sudoku puzzles belong to combinatorial problem (NP complete) and can be solved by using a variety of techniques/algorithms such as genetic algorithms, heuristics, integer programming, and many more. In this paper we proposed a new and different approach for solving Sudoku puzzles which is by modelling them as *block-world problems*. Block-world problem is known as one of the most famous planning domains in artificial intelligence. Given a set of blocks that are arranged in some vertical stacks/piles and some robots (possibly with different task capability), the problem is to change the initial arrangement into a new different arrangement. There have been several approaches for the modelling and solving the block-world problems ([13], [12]). Figure 2 gives an illustration of a block-world problem.

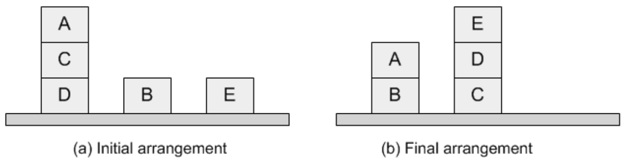


Fig. 2. An example of Block world problem.

The focus of our work is the modelling process and not the techniques/algorithms for solving the Sudoku. We take three simple techniques for solving Sudoku.

In this work, we use formal method approach for solving Sudoku puzzles. Referring to formal methods approach for system development, generally there are two important steps in developing systems, which are modeling and verification. A model (or specification) is a description of the behaviors of the system and usually written in a particular specification language. Verification is a process of proving the correctness of the models. Verification can also be viewed as a process of showing whether a model satisfies certain targeted behaviors called properties or not. Regarding our problem definition, which is solving the Sudoku puzzles, our frame of mind is by having a specification which is a representation of a Sudoku puzzle, the solution finding process can be done through verification by proving the property that eventually the solution is found. In this work, we are only focus on the modelling step. we use Temporal Logic of Actions (TLA) as the specification language.

This paper is organized as follows. Section II describes briefly parameterized multi-agent systems, in particular the TLA general formula for writing a specification of parameterized systems. Section III explains the how to model Sudoku puzzles as block world problems. The specifications of our proposed models are given in Section IV. Section V discusses how each model works. Section VI gives some related work. Conclusions and future work are given in Section VII.

Cecilia Nugraheni and Luciana Abednego is with the Informatics Department, Parahyangan Catholic University, Bandung, Indonesia(e-mail: cheni@unpar.ac.id and luciana@unpar.ac.id).

II. Parameterized multi-agent systems

In this work, multi-agent systems are viewed as parameterized systems. We call this kind of systems "parameterized multi-agent systems". A parameterized system is a system consisting of a number of similar components (subsystems) that work together, the number of the components is expressed as input parameter of the system. Furthermore, we only consider a class of parameterized systems which are *interleaving* (at any time there is only one active component).

Let M denote a finite and non-empty set of agents running in the system. A parameterized multi-agent system can be described as a formula of the form:

$$parMAS \equiv \forall k \in M : Init(k) \wedge \Box[Next(k)]_{v[k]} \wedge L(k).$$

where

- $Init$ is a predicate describing the status of the initial conditions,
- $Next(k)$ is an action (transition function) which states the relation next-state of a process k,
- v is a function of the status of the states variables of the system, and
- $L(k)$ is a formula that states the condition *liveness* expected from the process of subsystems k.

III. From Block World-Problems to Sudoku Puzzles

Following [4], in this work we use the following definition for Sudoku puzzle.

Definition 1 Given an $n^2 \times n^2$ cells divided into $n \times n$ distinct subblocks, the aim of Sudoku puzzle is to fill each cell so that the following three criteria are met:

1) Each row of cells contains the integers 1 through to n^2 exactly once.
2) Each column of cells contains the integers 1 through to n^2 exactly once.
3) Each subblock contains the integers 1 through to n^2 exactly once.

In this paper we focus on $n = 3$.

In order to model Sudoku puzzles as block-world problems, the first step is to analyze the components of a block-world problem. The components of a block-world problem is a number of robots, a number of boxes and a table. All the boxes are on the table and initially they are arranged into a number of piles. The task of the robots is to change this initial arrangement of these piles into a particular arrangement called final arrangement. Each robot has a different task (capability). The first robot is only capable to take a box from a table and put it on another box, whereas the second robot is capable to take a box from a top of a pile and put it on the table. It is assumed that every time only one robot that can make a move. Figure 2 shows an example of block world problem.

Viewing this problem as a multi-agent system, we may say that the agents are the robots and the environment consists of a number of boxes and a table. In [12] we have modeled block-world problem as parameterized multi-agent systems. The main actions of this specification are $move(k)$ and $free(k)$ representing the task of the first and the second robot, respectively. The reader may consult [12] for more detailed explanation.

Now let's change the setting of block world problem as follows:

- The number of boxes on the table is 9×9.
- Each box has number on it, which is between 1 and 9.
- For every number, from 1 to 9, there are 9 boxes with each number.
- There is a special part of the table called board which consists of 3×3 grids. Each grid is called subblock and consists of 3×3 smaller grids called cells.
- Initially some of the boxes are already placed on those cells and the rest are outside the board. The boxes that are outside the board are organized in nine pile according to their numbers.

We also make some changes of the behaviour of the robots:

- The first robot is capable of taking a box that are outside the board and putting this box on a cell of the board so that the condition in Definition 1 are satisfied.
- The second robot is capable of taking a box from the board and putting it back on the pile according to its number.

By changing the setting of block-world problems as explained, the problem has turned into a Sudoku problem. Then we may say that Sudoku puzzle is a variant of block-world problems.

IV. Proposed Models

A. First Model

Now we will design a scenario how the system works, in particular the task and the schedule of the robots. For the first model, we take a *backtracking style*. Our approach can be described as follows:

- The first robot gets the first turn. It is responsible for filling in the board with boxes which are still outside the board. This is done by searching an empty cell of the board, finding a box that can be placed on that cell, and putting the box on the cell. The first robot will do it repeatedly, until one of two conditions is reached:
  - All the cells are filled with boxes (which means no more boxes outside the board). At this situation the first robot will report that the job is done.
  - It is unable to find a box that can be placed on an empty cell. If this happens, then it will report that there is a failure and give the turn to the second robot.

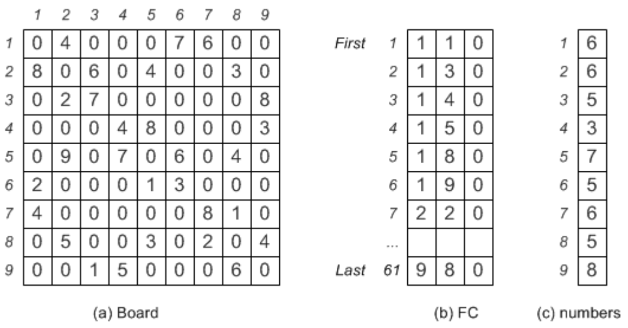
  For the process of searching an empty cell, the first robot will take the empty cell with the lowest position. We define the the lowest and highest position as $(1, 1)$ and $(9, 9)$, respectively. The first robot records all the empty and not empty cells from the beginning until the end of the process.
- The second robot will be active whenever a failure happens. It is responsible for taking a box from the board and put it back on its correspoding pile.

The reason for this action is based on the fact that the existence of this failure is because of the wrong choice made by the first robot. This implies that at least one of the boxes on the board is not the correct one and it must be taken away from the board. Since the first robot has no capability to take a box from a board, it gives a sign to the second robot to do this job. In our model, the second robot will take the last box put by the first robot. As consequence, the second robot needs information about the order of the box put on the board.

Now we ready to write the specification of our first model. As mentioned before, following [12], we will model Sudoku as parameterized multi-agent systems and will use the formula (1) for writing the specification for this problem.

Before that, we have to define the system's variables that will be used in the specification. There are seven variables used in the specification, namely:

1) $Board$
$Board$ represents the Sudoku's board. It is a 2-dimensional array of integers. Each element of Board represents a cell at a particular position according to the row and column. For example, if the value of $Board[3][5]$ is 6 then it means that the cell of third row and fifth column is already filled with a box with number 6. If the value of an element is equal 0 then it means that there is no box on that cell.
2) $numbers$
$numbers$ represents the piles of boxes that are not yet placed on the board. It is an array of integer. The indices, which are from 1 to 9, represent the box number and the value of the element represents the number of the boxes. For example, $numbers[5] = 3$ means that there are 3 boxes with number 5 that are still outside the board.
3) FC
FC is used to store the free/empty cells. A free cell is a board's cell without a box on it. There are two kind of free cells. The first kind is a cell that has never been used, whereas the second kind is a free cell that has been used already.
FC is organized as an array. Each element of FC is a triple of integers. The first and second elements of each triple represent the cell's position (row and column) on the board. The third element is used to record the box number that has been put on that cell. For example, an element $\langle 3, 5, 6 \rangle$ informs that the cell at the third row and the fifth column contains a box with number 6.
As explained, the both robots need information about empty cells for doing its task. FC is used for this purpose. It is assumed that initially FC contains the empty cell
4) $First$ and $Last$
$First$ and $Last$ is a non-negative integer representing the index of the first the last empty cell of the board, respectively. At the beginning, the value of $First$ is 1 and $Last$ is the number of the empty cells on the board. The value of $First$ will be changed (increased or decreased) depends on the actions taken by the robots. The value of $Last$, on the contrary, is fixed.
5) $isBackTrack$
$isBackTrack$ is a boolean variable representing the failure signal that is raised by the first robot as explained above. The first robot will set this variable to TRUE whenever it is failed to find a box to be put on a particular free cell. The second robot will only be active if $isBackTrack$ is TRUE. After doing its job, the second robot is responsible to reset this variable.
6) $isDone$
$isDone$ is a boolean indicating whether the finding solution process is finished or not.



	1	2	3	4	5	6	7	8	9
1	0	4	0	0	0	7	6	0	0
2	8	0	6	0	4	0	0	3	0
3	0	2	7	0	0	0	0	0	8
4	0	0	0	4	8	0	0	0	3
5	0	9	0	7	0	6	0	4	0
6	2	0	0	0	1	3	0	0	0
7	4	0	0	0	0	0	8	1	0
8	0	5	0	0	3	0	2	0	4
9	0	0	1	5	0	0	0	6	0

(a) Board

First	1	1	1	0
	2	1	3	0
	3	1	4	0
	4	1	5	0
	5	1	8	0
	6	1	9	0
	7	2	2	0
	...			
Last	61	9	8	0

(b) FC

1	6
2	6
3	5
4	3
5	7
6	5
7	6
8	5
9	8

(c) numbers

Fig. 3. Initial conditions of $Board, FC, numbers, First$, and $Last$.

It is assumed that at the beginning, the variables $Board$, $numbers$, and FC already contain values corresponding to the puzzle to be solved.

As illustration, the initial conditions/values of $Board$, FC, $numbers$, $First$, and $Last$ corresponding to the puzzle in Figure 1 are given in Figure 3.

The specification of our first model is given in Figure 4. The value of M represents the number of robots in the system, which in this case is 2. The main actions of the specification are $PutOn(k)$ and $TakeBack(k)$. Besides them, we also define some functions, namely:

1) $isEmpty(x, y)$
$isEmpty$ is a boolean function for checking whether a cell at the position (x, y) is empty or not.
2) $isSolved$
$isSolved$ is a boolean function which will return TRUE if the first robot put all the boxes on the board successfully, which means that there are no empty cells on the board.
3) $isNeighbor(i, j)$
$isNeighbor$ is a boolean function. This function is used to check whether two rows/column, i and j, are in the same subblock.
4) $isInRow(x, i)$
$isInRow(x, i)$ is a function for checking whether there is a box with number i on the row x.
5) $isInColumn(y, i)$
$isInColumn(y, i)$ is a function for checking whether there is a box with number i on the column y or not.
6) $isInSubBlock(x, y, i)$
$isInSubBlock(x, y, i)$ is a function for checking the existence of a box with number i in a subblock which

contains position (x, y).

7) $isValidPosition(x, y, i)$
$isValidPosition(x, y, i)$ is a function for checking whether a box with number i can be placed on a cell with position (x, y) with respect to the Sudoku's rules as stated in Definition 1.

The main actions of this specification are $PutOn(k)$ and $TakeBack(k)$. These actions can be taken only if the solution finding process is not yet finished ($\neg isDone \land \neg isSolved$).

Action $PutOn(k)$ can be taken only by a type 1 robot ($k = 1$). If there is a box with number i ($numbers[i] > 0$) that can be put on the first empty cell ($isValidPosition(FC[First][1],\ FC[First]\ [2])$ then the robot takes that box from its corresponding pile ($numbers' = [numbers \text{ EXCEPT } ![i] = numbers[i] - 1]$), puts the box on the empty cell (Board' = [Board EXCEPT ![FC[First][1]] [FC[First][2]] = i]), records which box put on that cell ($FC' = [FC \text{ EXCEPT } ![First][3]]$ and changes the first empty cell's pointer ($First' = First + 1$). The robot also determines whether the process should be terminated or not by checking the existence of boxes on the piles ($\forall p \in \{1..9\} : numbers[p] = 0$). If there is no such box, the robot will set the failure sign ($isBackTrack' = \text{TRUE}$). Notice that in choosing a box, the robot will pick a box with smallest number ($\forall j \in 1..i-1 : \neg isValidPosition(FC[First][1], FC[First][2], j)$) that has never been put on that cell ($i > FC[First][3]$).

Action $TakeBack(k)$ can be taken only by second type robots and only if the failure sign is set $isBackTrack = \text{TRUE}$. If it is not possible to do the backtrack ($First = 0$), the robot will terminate the solution searching process $isDone' = \text{TRUE}$. Otherwise, it will take back the last box put by the first robot ($Board' = [Board \text{ EXCEPT } ![FC[oF][1]][FC[oF][2]] = 0]$), put it back to its corresponding pile ($numbers' = [numbers \text{ EXCEPT } ![n] = numbers[m] + 1]$), and change the pointer of first empty cell ($First' = oF$).

We leave the values of $Board, FC, numbers$, and $Last$ empty in the specification. The values of those variables depend on the problem instance at hand.

B. Second Model

Differ from the first model, for the second model we use nine robots. Each robot is a first type robot, which is a robot whose task is to put a box on the board. We use the *fixed-point* principle for this model. The searching for the solution is done by making iterations until a termination condition is reached. In each iteration, every robot i tries to find an appropriate cell or a valid position for a box with number i. If the searching is success, then robot i puts a box with number i on it. After doing its job, the robot i will give the turn to the next robot.

After the robot 9 does its job, it will be decided whether a new iteration should be made or not. If in the last iteration, all the robots cannot find appropriate cells, in other word, the robots can not put any boxes on the board anymore, then the process is terminated.

In this model, we use the following definition for the valid position.

```
─────────────── module Model 1 ───────────────
$isEmpty(x, y) \equiv Board[x][y] = 0$
$isSolved \equiv \forall x, y : \neg isEmpty(x, y)$
$isNeighbor(i, j) \equiv \exists n : (i-1)\ \%\ 3 = n \land (j-1)\ \%\ 3 = n$
$isInRow(x, i) \equiv \exists y : Board[x][y] = i$
$isInColumn(y, i) \equiv \exists x : Board[x][y] = i$
$isInSubBlock(x, y, i) \equiv \exists j, k : \land \langle x, y \rangle \neq \langle j, k \rangle$
                                        $\land isNeighbor(x, j)$
                                        $\land isNeighbor(y, k)$
                                        $\land Board[j][k] = i$
$isValidPosition(x, y, i) \equiv \land isEmpty(x, y)$
                                 $\land \neg isInRow(x, i)$
                                 $\land \neg isInColumn(y, i)$
                                 $\land \neg isInSubBlock(x, y, i)$

$PutOn(k) \equiv$
$\land k = 1 \ \land \neg isDone \ \land \neg isSolved$
$\land$ IF $\exists i : \land i > FC[First][3] \land i \leq 9$
        $\land isValidPosition(FC[First][1], FC[First][2], i)$
        $\land numbers[i] > 0$
        $\land \forall j \in 1..i-1 :$
        $\land \neg isValidPosition(FC[First][1], FC[First][2], j)$
    THEN $\land Board' = [Board \text{ EXCEPT } ![FC[First][1]]$
                                              $[FC[First][2]] = i]$
         $\land FC' = [FC \text{ EXCEPT } ![First][3] = i]$
         $\land numbers' = [numbers \text{ EXCEPT } ![i] =$
                                              $numbers[i] - 1]$
         $\land First' = First + 1$
         $\land$ IF $\forall p \in \{1..9\} : numbers[p] = 0$
             THEN $\land isDone' = \text{TRUE}$
                  $\land$ UNCHANGED $\langle Last, isBackTrack \rangle$
             ELSE UNCHANGED $\langle Last, isDone, isBackTrack \rangle$
    ELSE $\land isBackTrack' = \text{TRUE}$
         $\land$ UNCHANGED $\langle Board, FC, numbers, First \rangle$
         $\land$ UNCHANGED $\langle Last, isDone \rangle$

$TakeBack(k) \equiv$
$\land k = 2 \ \land isBackTrack$
$\land \neg isDone \ \land \neg isSolved$
$\land isBackTrack' = \text{FALSE}$
$\land$ IF $First = 0$
    THEN $\land isDone' = \text{TRUE}$
         $\land$ UNCHANGED $\langle Board, FC, numbers, First, Last, \rangle$
    ELSE LET $oF = First - 1$
             $n = FC[First - 1][3]$
             $m = numbers[FC[First][3]]$
         IN  $\land Board' = [Board \text{ EXCEPT } ![FC[oF][1]]$
                                            $[FC[oF][2]] = 0]$
             $\land numbers' = [numbers \text{ EXCEPT } ![n] =$
                                            $numbers[m] + 1]$
             $\land First' = oF$
             $\land$ UNCHANGED $\langle FC, Last, isSolved \rangle$
$Init \equiv \land isBackTrack = \text{FALSE} \land \ isDone = \text{FALSE}$
        $\land isSolved = \text{FALSE} \land \ First = 0$
        $\land Last = ... \land \ Board = ...$
        $\land numbers = ... \land \ FC = ...$

$Next(k) \equiv PutOn(k) \lor \ TakeBack(k)$
$L(k) \equiv \text{WF}_v(PutOn(k)) \land \ \text{WF}_v(TakeBack(k))$
$v \equiv \langle Board, FC, numbers, First, Last, isDone, isBackTrack \rangle$

$Sudoku1 \equiv \land Init$
            $\land \Box[\exists k \in M : Next(k)]_v$
            $\land \forall k \in M : L(k)$
──────────────────────────────────────────────
```

Fig. 4. The specification of the first model.

Definition 2 A robot i will call a position (x, y) is valid for box with number i if the following conditions hold:

1) It is empty.
2) The row x does not contain any boxes with number i.
3) The column y does not contain any boxes with number i.
4) The subblock where (x, y) is located does not contain any boxes with number i.
5) For every other row r in the same subblock there is a box with number i.
6) For every other column c in the same subblock there is a box with number i.

The specification of the second model is given in Figure 5. The value M in this model is 9 and this time it does not represent the robot type but the robot identity. The system's variables used in the specification are $Board, numbers, turn, isDone$, and $success$. Variables $Board, numbers$, and $isDone$ are the same as the ones in the first specification. $turn$ is an integer variable used to manage the scheduling of the robots and $success$ is a boolean variable indicating the success of some robots in putting boxes at a certain position.

In this model, not like in the first model, the robots do not need any information about the empty/free cells. The searching for a valid position is done by observing the board directly. Each robot can freely choose any cell as long as it meets the conditions in Definition 2.

Some functions used in this specification that are not the same as or similar to the ones of the first model are $isInOtherRow(x, i)$, $isInOtherColumn(x, i)$, and $isValidPosition(x, y, i)$. These functions can be explained as follows:

1) $isInOtherRow(x, i)$
   $isInOtherRow(x, i)$ is a boolean function for checking the existence of a box with number i in two other rows in the same subblock with row x.
2) $isInOtherColumn(y, i)$
   $isInOtherColumn(y, i)$ is a boolean function for checking the existence of a box with number i in two other columns in the same subblock with column y.
3) $isValidPosition(x, y, i)$
   $isValidPosition(x, y, i)$ is a boolean function for checking whether the position (x, y) is a valid position for the box with number i according to the Definition 2.

The only main action in this specification is $PutOn(k)$. This action describes the working procedure of the robots. Robot k can activate $PutOn(k)$ only if it is in turn ($k = turn$) and as long as the searching process has been not yet terminated ($\neg isDone \wedge \neg isSolved$). If there is still a box with number k in its corresponding pile ($numbers[k] > 0$) and there is a valid position for k ($\exists x, y : isValidPosition(x, y, k)$), the robot takes the box ($numbers' = [numbers \text{ EXCEPT } ![k] = numbers[k] - 1]$), puts the box on that cell ($Board' = [Board \text{ EXCEPT } ![x][y] = k]$), and reports its success in doing its job ($success' = \text{TRUE}$). After that, the robot gives the turn to the next robot ($turn' = turn + 1$ or $turn' = 1$). The last robot is responsible to decide whether a new iteration should be made or not (IF $\neg$ $success$ THEN $isDone' = \text{TRUE}$).

```
module Model 2
$isSolved \equiv \forall x, y : Board[x][y] \neq 0$
$isNeighbor(i, j) \equiv \exists n : (i - 1) \% 3 = n \wedge (j - 1) \% 3 = n$
$isEmpty(x, y) \equiv Board[x][y] = 0$
$isInRow(x, i) \equiv \exists y : Board[x][y] = i$
$isInColumn(y, i) \equiv \exists x : Board[x][y] = i$
$isInSubBlock(x, y, i) \equiv \exists j, k : \wedge \langle x, y \rangle \neq \langle j, k \rangle$
                                   $\wedge isNeighbor(x, j)$
                                   $\wedge isNeighbor(y, k)$
                                   $\wedge Board[j][k] = i$
$isInOtherRow(r_1, r_2, x, y, i) \equiv$
$\wedge r_1 \neq r_2 \wedge r_1 \neq x \wedge r_2 \neq x$
$\wedge isNeighbor(r_1, x) \wedge isInRow(r_1, i)$
$\wedge isNeighbor(r_2, x) \wedge isInRow(r_2, i)$
$isInOtherColumn(c_1, c_2, x, y, i) \equiv$
$\wedge c_1 \neq c_2 \wedge c_1 \neq y \wedge c_2 \neq y$
$\wedge isNeighbor(c_1, y) \wedge isInColumn(c_1, i)$
$\wedge isNeighbor(c_2, y) \wedge isInColumn(c_2, i)$
$isValidPosition(x, y, i) \equiv$
$\wedge isEmpty(x, y) \wedge \neg isInRow(x, i)$
$\wedge \neg isInColumn(y, i) \wedge \neg isInSubBlock(x, y, i)$
$\wedge \exists r_1, r_2 : isInOtherRow(r_1, r_2, x, y, i)$
$\wedge \exists c_1, c_2 : isInOtherColumn(c_1, c_2, x, y, i)$

$PutOn(k) \equiv$
$\wedge \neg k = turn \wedge \neg isDone \wedge \neg isSolved$
$\wedge$ IF $numbers[k] > 0 \wedge \exists x, y : isValidPosition(x, y, k)$
    THEN $\wedge numbers' = [numbers$ EXCEPT $![k] = numbers[k] - 1]$
         $\wedge Board' = [Board$ EXCEPT $![x][y] = k]$
         $\wedge success' = $ TRUE
    ELSE UNCHANGED $\langle Board, numbers \rangle$
$\wedge$ IF $turn < 9$ THEN $\wedge turn' = turn + 1$
                    $\wedge$ UNCHANGED $\langle success, isDone \rangle$
               ELSE $\wedge$ IF $\neg success$ THEN $\wedge isDone' = $ TRUE
                                         $\wedge success' = success$
                                         $\wedge turn' = turn$
                                    ELSE $\wedge turn' = 1$
                                         $\wedge success' = $ FALSE
                                         $\wedge$ UNCHANGED $\langle isDone \rangle$

$v \equiv \langle Board, numbers, turn, success, isDone \rangle$
$Init \equiv \wedge turn = 1 \wedge isDone = $ FALSE $\wedge success = $ FALSE
        $\wedge Board = ...$
        $\wedge numbers = ...$
$Next(k) \equiv PutOn(k)$

$L(k) \equiv \mathrm{WF}_v(PutOn(k))$
$Sudoku2 \equiv \wedge Init$
            $\wedge \Box[\exists k \in M : Next(k)]_v$
            $\wedge \forall k \in M : L(k)$
```

Fig. 5. Specification of the second model.

C. Third Model

Basically, the third model is similar to the second one. The difference is on the definition of a valid position.

Definition 3 A robot i will call a cell at (x, y) a valid position if all the following conditions hold:

1) It is empty.
2) The row x does not contain any boxes with number i.

3) The column y does not contain any boxes with number i.
4) The subblock where (x, y) is located does not contain any boxes with number i.
5) For every other row r in the same subblock there is a box with number i or the cell at the position (r, y) is not empty.
6) For every other column c in the same subblock there is a box with number i or the cell at the position (x, c) is not empty.

The specification for the third model is given in Figure 6. In principle, this specification is very similar to the one of the second specification. The difference is only on the definition of function $isInOtherRow(r_1, r_2, x, y, i)$ and $isInOtherColumn(c_1, c_2, x, y, i)$. The difference between these actions can be explained by using the Sudoku puzzle in Figure 1 (a). Using model 2 and 3, a robot 4 can put a box with number 4 at the location $(3, 7)$. However with model 2 the robot 2 cannot put a box with number 2 at the location $(7, 3)$. This is because there is no box with number 2 in the row 9. Model 3 enables the robot 2 to put the box on position $(7, 3)$ since the position $(9, 3)$ is not empty so that the position $(7, 3)$ is the only candidate of a valid position for 2 in that subblock.

V. Discussion

We now make a brief analysis of the three proposed models. Assuming that the puzzles always have solutions, like the one in Figure 1, the first model will always find one the solutions. This does not hold for the second and the third models.

Figure 7 shows 3 first iterations for the second model. In the first iteration, the system can put 2 box on the board, which are 4, and 8. In the second and third iteration, only robot 4 that can do its job. After third iteration, the process is terminated.

Figure 8 shows 3 first iterations. In the first iteration, the system can put 6 box on the board, which are $2, 3, 4, 5, 6$, and 8. In the second iteration, only two robots can put their boxes, which are 4 and 8. In the third iteration, three more boxes can be put on the board, which are $3, 4$, and 8. Continuing the iterations, all the empty cells can be filled in 13 iterations. Thus, the puzzle can be solved by the third model.

We may say that the third model is an improvement of the second model. It is clear that in general the third model will outperform the second model.

VI. Related Work

There have been many work dedicated to Sudoku puzzles. Most of them are related to the solution finding problem. Many approaches have been proposed , such as genetic algorithms ([6], [7]), simulated annealing ([4], [8]), neural networks ([14], [9]), integer programming ([3], [1]), Particle Swarm Optimisation ([11], [10]), SAT ([5]), and so on.

Besides, there are also plenty sites in the internet about heuristics for solving Sudoku puzzle, for example [2]. The heuristics used in this model work (for the second and third model) are simple heuristics that can be viewed as a part of the heuristics found in [2].

```
module Model 3
$isSolved \equiv \forall x, y : Board[x][y] \neq 0$
$isNeighbor(i, j) \equiv \exists n : (i - 1) \% 3 = n \land (j - 1) \% 3 = n$
$isEmpty(x, y) \equiv Board[x][y] = 0$
$isInRow(x, i) \equiv \exists y : Board[x][y] = i$
$isInColumn(y, i) \equiv \exists x : Board[x][y] = i$
$isInSubBlock(x, y, i) \equiv \exists j, k : \land \langle x, y \rangle \neq \langle j, k \rangle$
                                      $\land isNeighbor(x, j)$
                                      $\land isNeighbor(y, k)$
                                      $\land Board[j][k] = i$
$isInOtherRow(r_1, r_2, x, y, i) \equiv$
$\land r_1 \neq r_2 \land r_1 \neq x \land r_2 \neq x$
$\land isNeighbor(r_1, x) \land isInRow(r_1, i) \lor \neg isEmpty(r_1, y)$
$\land isNeighbor(r_2, x) \land isInRow(r_2, i) \lor \neg isEmpty(r_2, y)$
$isInOtherColumn(c_1, c_2, x, y, i) \equiv$
$\land c_1 \neq c_2 \land c_1 \neq y \land c_2 \neq y$
$\land isNeighbor(c_1, y) \land isInColumn(c_1, i) \lor \neg isEmpty(x, c_1)$
$\land isNeighbor(c_2, y) \land isInColumn(c_2, i) \lor \neg isEmpty(x, c_2)$

$isValidPosition(x, y, i) \equiv$
$\land isEmpty(x, y) \land \neg isInRow(x, i)$
$\land \neg isInColumn(y, i) \land \neg isInSubBlock(x, y, i)$
$\land \exists r_1, r_2 : isInOtherRow(r_1, r_2, x, y, i)$
$\land \exists c_1, c_2 : isInOtherColumn(c_1, c_2, x, y, i)$

$PutOn(k) \equiv$
$\land \neg k = turn \land \neg isDone \land \neg isSolved$
$\land$ IF $numbers[k] > 0 \ \land \exists x, y : isValidPosition(x, y, k)$
    THEN $\land numbers' = [numbers$ EXCEPT $![k] = numbers[k] - 1]$
         $\land Board' = [Board$ EXCEPT $![x][y] = k]$
         $\land success' =$ TRUE
    ELSE UNCHANGED $\langle Board, numbers \rangle$
$\land$ IF $turn < 9$ THEN $\land turn' = turn + 1$
                     $\land$ UNCHANGED $\langle success, isDone \rangle$
              ELSE $\land$ IF $\neg success$ THEN $\land isDone' =$ TRUE
                                        $\land success' = success$
                                        $\land turn' = turn$
                                  ELSE $\land turn' = 1$
                                        $\land success' =$ FALSE
                                        $\land$ UNCHANGED $\langle isDone \rangle$

$v \equiv \langle Board, numbers, turn, success, isDone \rangle$
$Init \equiv \land turn = 1 \land isDone =$ FALSE $\land success =$ FALSE
         $\land Board = ...$
         $\land numbers = ...$
$Next(k) \equiv PutOn(k)$

$L(k) \equiv \mathrm{WF}_v(PutOn(k))$
$Sudoku3 \equiv \land Init$
            $\land \Box[\exists k \in M : Next(k)]_v$
            $\land \forall k \in M : L(k)$
```

Fig. 6. Specification of the third model.

VII. Conclusion and Future Work

We have shown that Sudoku problem can be modeled as block-world problems by changing the setting of block-world problems. In this work, three models for the Sudoku problem have been developed. The first model uses a *backtracking* technique and the rest models are based on simple heuristics and *fixed-point* principle.

We have formally written our models in Temporal Logic of Actions. The correctness of these models are not yet proved. In our next work we will do the verification by using model

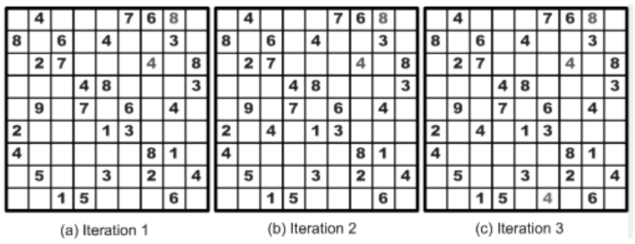


Fig. 7. First iterations of the second model.

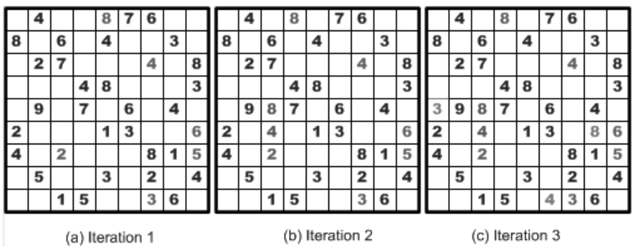


Fig. 8. First iterations of the third model.

checking or another verification technique. Following [12] we have modeled the Sudoku puzzles as multi-agent systems. However, in this paper, we have not yet explored the issues related to multi-agent systems. Currently, we are developing a program that implement the models. In this implementation we consider more about these issues. Furthermore, using this program and a set of Sudoku puzzles, an experiment will be conducted to analysis the performance of each model.

REFERENCES

[1] A. Bartlett, et.al. *An Integer Programming Model for the Sudoku Problem*, The Journal of Online Mathematics and Its Applications: Volume 8, Issue May, 2008.

[2] A. Gupta, *How to solve Su Doku: tips*, 2006, available at http://theory.tifr.res.in/~sgupta/sudoku/algo.html.

[3] T. Koch, *Rapid Mathematical Programming or How to Solve Sudoku Puzzles in a few Seconds*, Konrad-Zuse-Zentrum fur Informationstechnik Berlin, ZIB Report 05-51.

[4] R. Lewis, *Metaheuristics can solve Sudoku puzzles*, Journal of Heuristics, Vol. 13, 2007, pp. 387-401.

[5] I. Lynce & J. Ouaknine. *Sudoku as a SAT problem*, Proc. of the 9th Symposium on Artificial Intelligence and Mathematics, 2006.

[6] T. Mantere. *Solving, rating and generating Sudoku puzzles with GA.*, Proc. of IEEE Congress on of Evolutionary Computation, 2007. CEC 2007, Singapore.

[7] X.Q. Deng & Y.D. Li *A novel hybrid genetic algorithm for solving Sudoku puzzle.*, Optimization Letters, February 2013, Volume 7, Issue 2, pp 241-257, Springer.

[8] P. Malakonakis, et.al. *An FPGA-based Sudoku Solver based on Simulated Annealing methods.*, Proc. of International Conference on Field-Programmable Technology, 2009. FPT 2009.

[9] V. Mladenov, et.al., *Solving Sudoku Puzzles by using Hopfield Neural Networks*, Proc. of ICACM'11 Proceedings of the 2011 international conference on Applied & computational mathematics, pp.174-179 World Scientific and Engineering Academy and Society (WSEAS) Stevens Point, Wisconsin, USA, 2011.

[10] J. Monk, et.al. *Solving Sudoku using Particle Swarm Optimization on CUDA*, Proc. of The 2012 International Conference on Parallel and Distributed Processing Techniques and Applications, 2012,.

[11] A. Moraglio & J. Togelius, *Geometric Particle Swarm Optimization for the Sudoku Puzzle*, Proc. of Conference in Genetic and Evolutionary Computation, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007. ACM 2007.

[12] C.E. Nugraheni. *Formal Verification of Parameterized Multi-agent Systems Using Predicate Diagrams**. Proc. of COMPUTATION TOOLS 2011: The Second International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking, Rome, 2011.

[13] T. Taibi. *Formal specification and validation of multi-agent behaviour using TLA+ and TLC model checker*. Int. J. Artificial Intelligence and Soft Computing, Vol. 1, No. 1, pp. 99-113, 2008.

[14] T.-W. Yue & Z.-C. Lee, *Sudoku Solver by Qťron Neural Networks*, Proc. of International Conference in Intelligent Computing 2006, ICIC2006, LNCS 4113, pp.943-952, 2006, Springer-Verlag, Berlin Heidelberg 2006.