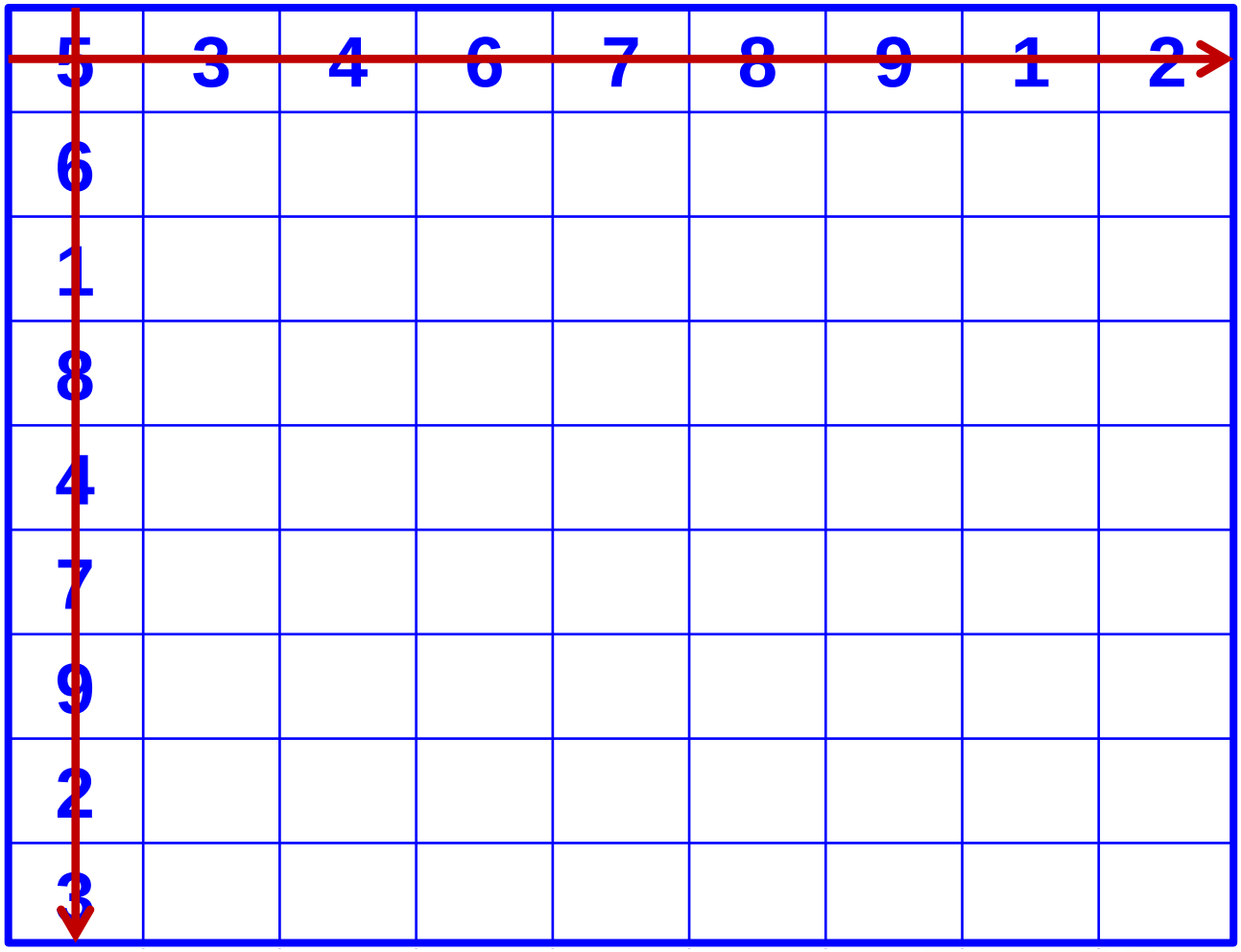
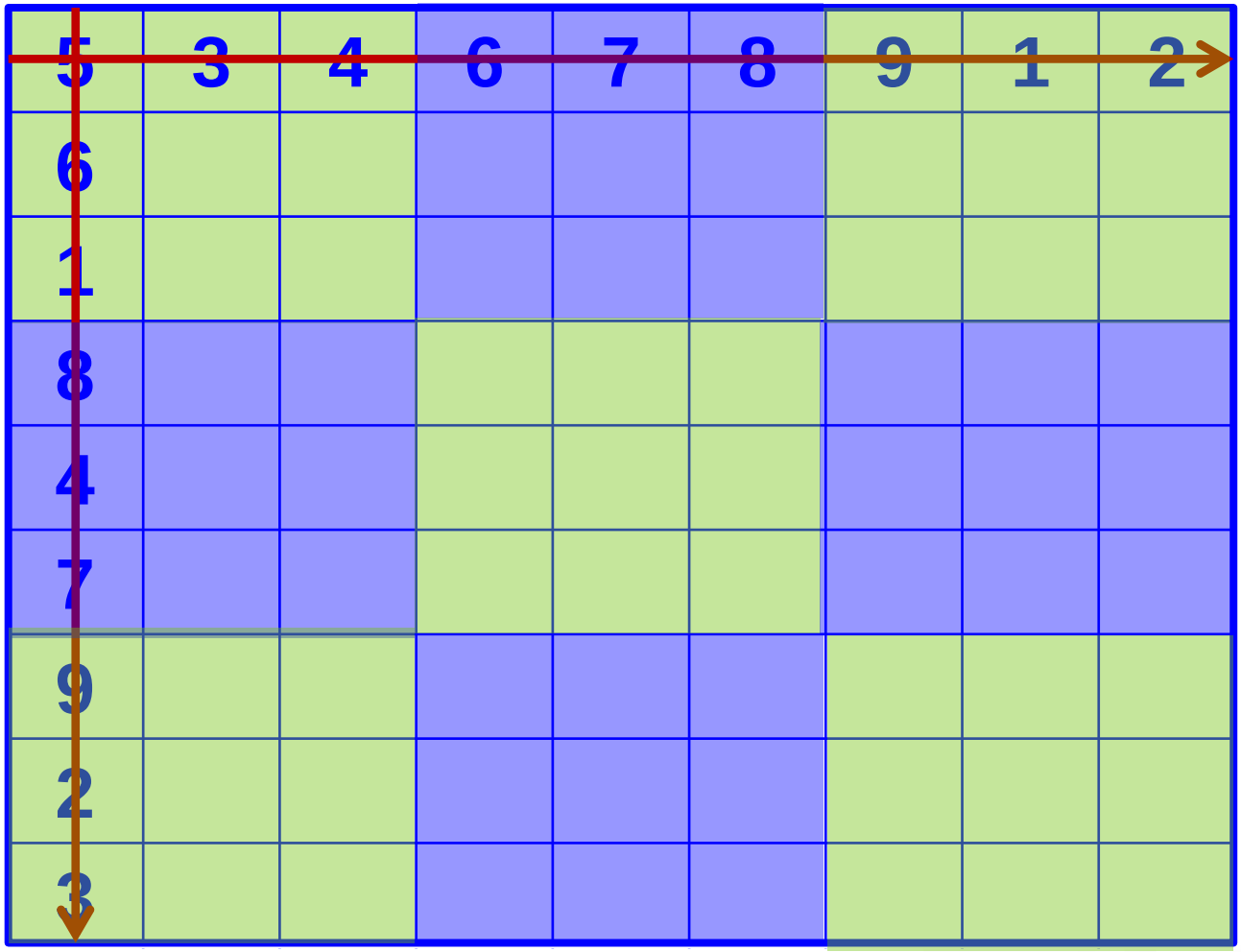
Yet Another Sudoku Solver: PROC FCMP

2012 Orlando Florida
April 22-25, 2012

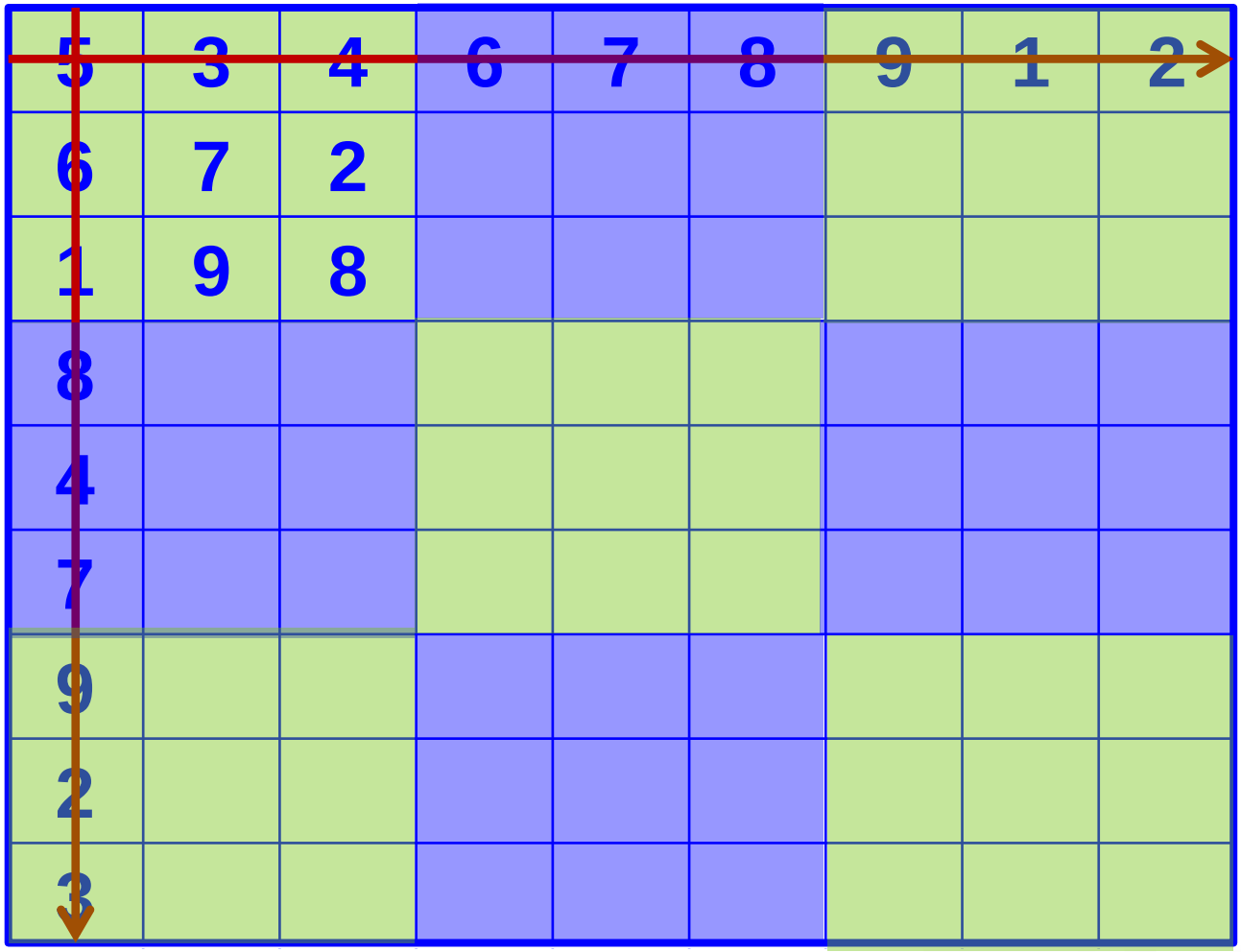
Sudoku: the rules



Sudoku: the rules



Sudoku: the rules



Sudoku: the rules

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

Sudoku: the rules

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

SAS-based Sudoku solver solutions have been proposed using:

a fixed path with arrays and macros

bootstrapping

proc sql

linear optimization (proc assign and proc lp)

constraint programming (proc clp)

candidate elimination

datastep programming

problems/limitations of previous approaches
(i.e., why we HAD to create yet another Sudoku Solver)

6. Some can not solve difficult puzzles
5. Some have not published all needed code
4. Some require specialized SAS components
3. Base SAS is not a matrix-oriented language
2. One of the paper's authors was bored one day

problems/limitations of previous approaches
(i.e., why we HAD to create yet another Sudoku Solver)

a void MUST be filled

No one has proposed
a proc fcmp solution

with PROC FCMP you can create:

- functions that convert SAS datasets into the matrices the data actually represent
- functions that analyze the matrices directly
- bi-directional recursive functions
- analyses that apply deterministic back-tracking logic

a proc fcmp solution:

```
proc fcmp;

 function parseProblem(p $);
  length puzzle $82;
  puzzle=p || ' ';
  array problem[9,9] /nosymbols;
  k=1;
  do i=1 to 9;
   do j=1 to 9;
    problem[i,j]=input(substr(puzzle,k,1),1.);
     k+1;
   end;
  end;
  rc=write_array('sudoku',problem);
  return(rc);
 endsub;
```

a proc fcmp solution (continued):

```
function solveProblem(u,p[*,*]);
  if u then do;
   if solveForward(1,1,p) then do;
    array f[9,9] /nosymbols;
    rc+read_array('sudoku_f',f);
   end;
   if solveReverse(9,9,p) then do;
    array r[9,9] /nosymbols;
    rc+read_array('sudoku_r',r);
   end;
    m=0;
    if rc=0 then
    do i=1 to 9;
     do j=1 to 9;
       z=(f[i,j]=r[i,j]);
      m+z;
     end;  end;  end;
```

a proc fcmp solution (continued):

```
else do;
    r1=0; r9=0;
     do i=1 to 9;
       r1+ifn(p[i,1]=0,1,0);  r9+ifn(p[i,9]=0,1,0);
     end;
     if r1<r9 then do;
     if solveForward(1,1,p) then do;
      array f[9,9] /nosymbols;
       m=81+read_array('sudoku_f',f);
    end;  end;
    else do;
     if solveReverse(9,9,p) then do;
      array f[9,9] /nosymbols;
      m=81+read_array('sudoku_r',f);
  end;  end;  end;
  if m=81 then z=writeMatrix(f);
  return(ifn(m=81,1,0));
 endsub;
```

a proc fcmp solution (continued):

```
function solveForward(_i,_j,c[*,*]);
 array cells[9,9] /nosymbols;
 do a=1 to 9;  do b=1 to 9;
   cells[a,b]=c[a,b];
 end;  end;
 i=_i; j=_j;
 if i>9 then do;
  i=1; j+1;
  if(j>9) then do;
   rc=write_array('sudoku_f',cells);   return(1);
 end;  end;
 if cells[i,j] ne 0 then return(solveForward(i+1,j,cells));
 do val=1 to 9;
  if legal(i,j,val,cells) then do;
   cells[i,j]=val;
   if (solveForward(i+1,j,cells)) then return(1);
 end;  end;  cells[i,j]=0;  return(0);
endsub;
```

a proc fcmp solution (continued):

```
function solveReverse(_i,_j,c[*,*]);
 array cells[9,9] /nosymbols;
 do a=1 to 9;  do b=1 to 9;
   cells[a,b]=c[a,b];
 end;  end;   i=_i; j=_j;
 if i<1 then do;
  i=9; j=j-1;
  if(j<1) then do;
   rc=write_array('sudoku_r',cells);   return(1);
  end;  end;
 if cells[i,j] ne 0 then return(solveReverse(i-1,j,cells));
 do val=1 to 9;
  if legal(i,j,val,cells) then do;
   cells[i,j]=val;
   if (solveReverse(i-1,j,cells)) then return(1);
 end;  end;   cells[i,j]=0;   return(0);
endsub;
```

a proc fcmp solution (continued):

```
function legal(i,j,val,cells[*,*]);
do k=1 to 9; *scan row;
 if val=cells[k,j] then  return(0);
end;
do k=1 to 9;
 if val=cells[i,k] then return(0);
end;
roffset=i-mod(i-1,3);
coffset=j-mod(j-1,3);
do k=0 to 2; *scan box;
 do m=0 to 2;
  if val=cells[roffset+k,coffset+m] then return(0);
 end;
end;
return(1);
endsub;
```

a proc fcmp solution (continued):

```
function writeMatrix(solution[*,*]);
  put "   ---------------------";
  do i=1 to 9;
    put @2 '|' @;
    h=4;
    do j=1 to 9;
      x=ifc(solution[i,j]=0,' ',put(solution[i,j],1.));
      put @h x $2. @; h+2;
      if mod(j,3)=0 then do; put @h "| " @; h+2; end;
    end;
    put /;
    if mod(i,3)=0 then put @1 "   ---------------------";
  end;
  return(rc);
endsub;
```

a proc fcmp solution (continued):

```
array args[11] $81 (
  '1000070900300200080096005000053009000100800026000040003000000100400000070070003000'
  '0000000700600100040034002008000030500029007000400800090200600070001009007000080600'
  '1005004000090300000700080050010000308006005000900070080040200102008006000000010020'
  '0800000010070040206003007000020090001000600080304000000017006000900080050000000400'
  '1004008000400300090090060500503000000000016000000700020040109007008000040200040800'
  '0050097000600000201008000060107000040070600306000032000000060400900501008001000020'
  '6000002000900010050080300400000020015006009000070900000700030020004005000060700800'
  '1000000600001000030050029000090010007000400800305000025004000060080600700700050000'
  '0000100040302000006000080900070600059000050800008004000409001007000020400050300070'
  '4000600700000006000300020017000085000104000000209500000000007050091000300030400800'
  '0053000008000000200700105004000053000100700060032000800605000090040000300000097000'
);
```

a proc fcmp solution (continued):

```
array test[11] (0 0 0 0 0 0 0 0 0 0 0 );
do i=1 to dim(args);
 rc=parseProblem(args[i]);
 array problem[9,9] /nosymbols;
 rc=read_array('sudoku',problem);
 put 'Problem= ' args[i];
 x=writeMatrix(problem);
  _time=time();
 if solveProblem(test[i],problem)=0 then
   put 'No Unique Solution Found';
 _diff=time()-_time;
 put 'Time Elapsed: ' _diff best. 'seconds';
 put _page_;
end;
run;
```

All of the code and this Powerpoint can be found at:

http://www.sascommunity.org/wiki/SAS_Global_Forum_2012_Presentations

Notes:

1. The code may updated if improvements are made
2. The code in the paper has already been improved
3. The code NOW includes a Sudoku problem generator

The code:

- includes examples of how to create functions with PROC FCMP
- can solve most and possibly all problems
- only requires base SAS
- includes bi-directional recursive functions
- includes deterministic back-tracking logic
- is faster for the most difficult problems

Your comments and questions are valued and encouraged

SAS® GLOBAL FORUM

2012 Orlando Florida

Matthew Kastin
i-behavior, Inc., Louisville, Colorado
email: matthew.kastin@gmail.com

Arthur Tabachneck, Ph.D.
myQNA, Inc., Thornhill, Ontario
e-mail: atabachneck@gmail.com