APLAI Assignment 2013-2014

March 5, 2014

1 Introduction

The APLAI assignment has 3 parts. To pass for this course you have to do Task 1 and Task 2. Task 3 is optional.

1.1 Some practicalities

- You work in groups of 2: the 2014 groups are listed on Toledo (wiki of APLAI).
- Due date: Wednesday 04/06/2014.
- You email your report (a pdf file), programs and README file to gerda.janssens@cs.kuleuven.be.
- As this assignment is part of the evaluation of this course, the Examination Rules of the KULeuven are applicable. During the exam period, there is an oral discussion for each group on 18/06/2014 or on 25/06/2014. Only if these dates are not possible, 27/06/2014 will be scheduled as an additional date. Further arrangements via Toledo!
- Follow the rules of academic integrity:
  - Write your own solutions, programs and text. Run your own experiments. When receiving assistance, make sure it consists of general advice that does not cross the boundary into having someone else write the actual code. It is fine to discuss ideas and strategies, but be careful to write your programs on your own. Do not give your code to any other student who asks for it and do not ask anyone for a copy of their code. Similarly, do not discuss algorithmic strategies to such an extent that you end up turning in exactly the same code.
  - Use a scientific approach and mention your sources.
- Some more information about the report:
  - It contains the answers for Tasks 1 and 2, and optionally Task 3.
  - It has an introduction and conclusion. In the conclusion you give a critical reflection on your work. What are the strong points? and the weak points? What are the lessons learned?

  - When you run your experiments, do not just give the time and/or search results, but try to interpret the results.
  - You can include some code fragments in your report to explain your approach. If you do, make a good selection. Do not include the complete code in the report as the code is in the program files.
  - Add an appendix that reports on the workload of the project and on how you divided and allocated the tasks in this project.
  - A typical report (for Task 1 and Task 2) consists of 20 to 25 pages in total.

1.2 APLAI Systems

The APLAI WIKI page contains information about the installation and use of the ECLiPSe, CHR and Jess systems.

THE APLAI WIKI page also contains additional ECLiPSe and CHR exercises.

2 Task 1: Sudoku

2.1 Task 1.A Discussion

Discuss how Sudoku can be solved by the 3 different systems (ECLiPSe, CHR and Jess). Note that there can be more than one way to solve Sudoku within one system! Explain in detail:

- which aspects of the systems are useful and why;
- which aspects are not useful and why not;
- possible alternatives, advantages/disadvantages

2.2 Task 1.B Viewpoints and Programs

The classical viewpoint for Sudoku states that all numbers in a row must be different, that all numbers in a column must be different, and that all numbers in a block must be different.

- Give a **different** viewpoint.
- What are the criteria you would use to judge whether a viewpoint is a good one or not?
- Can you define **channeling** constraints between your alternative and the classical viewpoint?
- For 2 out of the 3 systems: program the 2 viewpoints and if possible the channeling between the 2 viewpoints.

Motivate your choices/decisions.

2.3 Task 1.C Experiments

Run experiments with your programs, for the different viewpoints and possibly channeling constraints, and discuss the obtained results (e.g. w.r.t. run-times and search behaviour) in a scientific way.

On Toledo you find a set of Sudoko puzzles (see `sudex_toledo.pl`). You should include them in your experiments, using the same names to refer to the puzzles, but you can also include your own favorite Sudoku puzzles!

- Give the results (e.g. timings and number of backtracks) for the (given) Sudoko puzzles and explain them.
- Do you get different results for your 2 implementations? when using different viewpoints? using channeling? Why?
- What is/are the most difficult puzzle(s) for your programs? Can you explain why this is the case?
- What are your conclusions?

3 Task 2: Slitherlink

3.1 Problem definition

Slitherlink is a Japanese game invented by Nikoli (of Sudoku fame). The game is played on a rectangular grid, where each cell is either empty, or contains a number between 0 and 3. The grid is made up of edges and each cell is surrounded by four of these. The player's task is to draw a single cycle without crossings along these edges. A number in a cell determines how many of the four surrounding edges have to be used. An example puzzle, and its solution, is shown in Figure 1.

We assume that all well-formed Slitherlink puzzles have a unique solution.

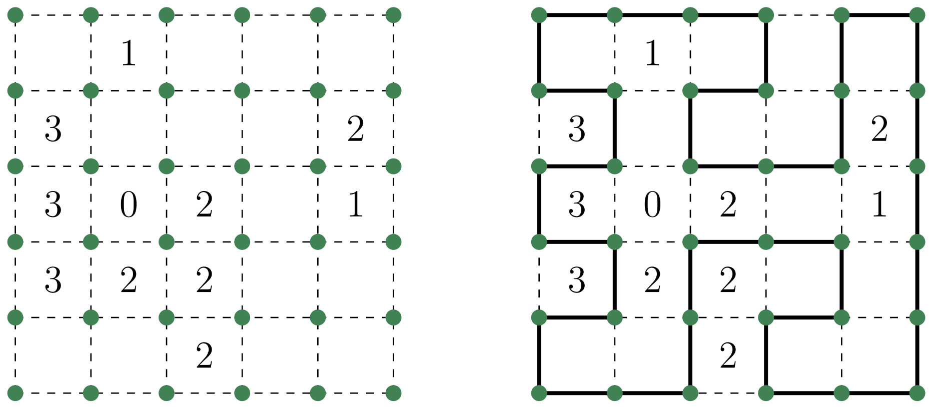


Figure 1: An example Slitherlink puzzle and its solution.

3.2 Example input and output

A program for solving this problem in ECLiPSe would be called with an input of this kind:

```
  slither(NRows,NCols,ListOfNumberedCells,ListOfEdges).
```

For example, for the puzzle in Figure 1 the input would be

```
slither(5,5,
   [             cell(1,2,1),
     cell(2,1,3),                                cell(2,5,2),
     cell(3,1,3), cell(3,2,0), cell(3,3,2),       cell(3,5,1),
     cell(4,1,3), cell(4,2,2), cell(4,3,2),
                               cell(5,3,2)
   ], Solution).
```

The result would be a list of values 0/1 where each number indicates whether the corresponding edge is part of the cycle or not. The horizontal edges come first, followed by the vertical edges, from left-to-right, top-to-bottom.

```
[ 1,1,1,0,1,    % Above row 1
  1,0,1,0,0,    % Above row 2
  1,0,1,1,0,    % Above row 3
  1,0,1,1,0,    % Above row 4
  1,0,0,1,0,    % Above row 5
  1,1,0,1,1,    % Below row 5
  1,0,0,1,1,1,  % Row 1
  0,1,1,0,1,1,  % Row 2
  1,0,0,0,0,1,  % Row 3
  0,1,1,0,1,1,  % Row 4
  1,0,1,1,0,1 ] % Row 5
```

The report “A rule-based approach to the puzzle of Slither Link”[1] gives some useful insights on how to represent and model this puzzle. (But be careful, because this paper does not model the constraint that the edges should form a single cycle.)

At the end of this assignment you can find some useful code you can use in your solution (see file `basic_slither.pl.txt` on Toledo).

3.3 Tasks

3.3.1 Task 2.A: Implementation in ECLiPSe

Implement a solver for the problem of Slitherlink as defined above in ECLiPSe.

Address the following questions:

- How do you represent the puzzle and the decision variables?
- How do you represent the constraints?

[1] `http://www.cs.kent.ac.uk/pubs/ug/2005/co620/slither/report.pdf`

- You can enforce the single cycle constraint as a post-processing step. Is there a smarter way? Could the `circuit`[2] constraint help? Why (not)? (It is sufficient to implement the single cycle constraint in post processing.)
- Would symmetry breaking speed up solving for this puzzle? If so, how would do this?
- Experiment with the built-in predicate `search/6`. What is the effect of different options?

Motivate your decisions (viewpoint, constraints, search strategy, design choices and any decisions for implementing additional predicates).

3.3.2 Task 2.B: Slitherlink in CHR/Jess

In order to program this problem in CHR or Jess, you will have to encode more things yourself such as propagation and search.

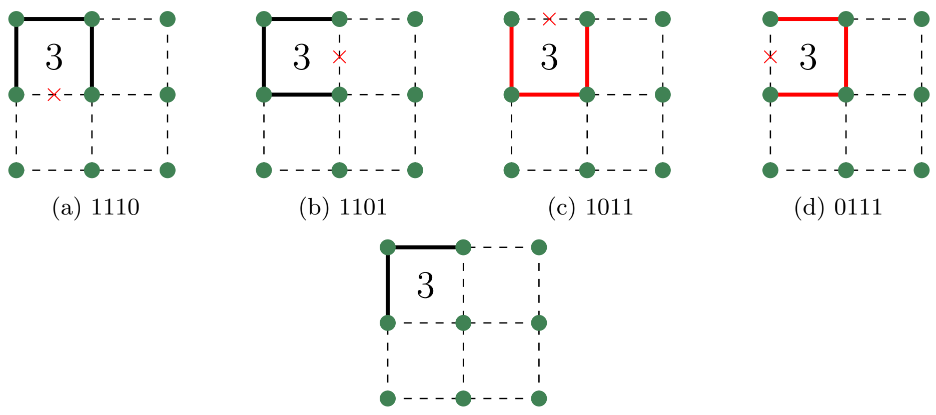
Your report should at least describe the following steps:

1. Choose and explain a suitable representation of the puzzle.
2. Choose and explain a suitable representation of the constraints (see below for some tips).
3. Implement a basic version of your program which uses depth-first search to compute a correct solution. Describe how you deal with constraint propagation, and what kind(s) of propagation you support.
4. Propose some additional constraints that may speed up solving, and implement a few of them (at least 2). Describe the effect of these additional constraints on the performance of your implementation.
5. Implement the alternative search algorithm explained below. Describe the effect of this search algorithm and compare it with depth-first search. How does the alternative search behave when the puzzle has an alternative solution? (You don't need to support this case.)

Run experiments with your CHR/Jess programs and discuss the results. The Toledo file `APLAI_slither_puzzles.ecl` contains our benchmark puzzles. Having basic versions of constraint propagation and search, your programs might not be able to deal with some of the benchmark puzzles. If this is the case, add you own examples/instances that illustrate the strong and/or weak points of your program versions.

Hint: single cycle constraint The constraint that states that the final solution should contain a single cycle can be implemented by maintaining line segments `segment(P1,P2)` where P1 and P2 are points in the grid. Whenever an edge is added, the segments should be updated. At the end there should be exactly one segment `segment(P,P)`.

[2] `http://eclipseclp.org/doc/bips/lib/gfd/circuit-1.html`



(a) 1110 (b) 1101 (c) 1011 (d) 0111

(e) Result: 11??

Figure 2: Illustration of alternative search.

Hint: one-hop breadth-first search Instead of depth-first search, we can also use a custom search strategy which focusses on exploring all alternatives for each individual constraint at-a-time. Assume for example that we have a 3 in the top-left corner of the grid as shown in Figure 2.

For this cell we have the constraint $v_{11} + h_{11} + v_{12} + h_{21} = 3$. There are four possible assignments to $(v_{11}, h_{11}, v_{12}, h_{21})$ that satisfy this constraint: 1110, 1101, 1011, 0111. However, the last two assignments are not valid because they cause a dead end in the upper left grid point. This means that in all valid assignments, that is (a) and (b), v_{11} and h_{11} should be 1.

The basic idea of this search strategy is thus to try out all possibilities for a single constraint (e.g. a cell), propagate the constraints, and look for common assignments in all of them. In CHR, this can be implemented using `findall/3`:

```
findall(Choice, ( choose(Choice), propagate(Choice) ), Choices),
   analyze_choices(Choices,FinalChoice),
   propagate(FinalChoice),
   continue_search.
```

The predicates `choose`, `propagate`, and `analyze_choices` are just examples. The actual predicates can (and will) look very different.

General tips:

- CHR and ECLiPSe are both based on Prolog. You may be able to reuse some code between both tasks.
- For CHR, use the SWI-Prolog version and not the version in ECLiPSe.
- There is some useful code in the appendix at the end of this text (and in the file `basic_slither.pl.txt`).
- If your CHR program makes SWI-Prolog run out of memory (e.g. Global Stack), restart the SWI system. Also after reloading your CHR program file a number of times, it is a good idea to restart SWI anyway.

- The file with slither puzzles can be loaded into SWI by the Prolog query ?-['APLAI_slither_puzzles.ecl'].
- To give you an idea of what our slither programs currently can do:
  - Our basic ECLiPSe program solves puzzles a10 and b10 in a few seconds, but does not find an answer for a15 or b15 with a timeout of 30 minutes.
  - Our advanced CHR solver (using `swipl -O -nodebug` and modes and types declarations for some of the chr_constraint declarations) solves all the benchmarks puzzles: the hardest puzzle is `big` which takes `95,117,558 inferences, 18.066 CPU in 18.207 seconds (99% CPU, 5265135 Lips)`.

4 Task 3

There are 3 alternatives for this optional task.

1. Study your own problem in detail.
   - Describe your own problem to be solved.
   - Why did you take this problem?
   - Why do you think it can benefit from the APLAI methods?
   - Write and discuss your 2 programs in detail.
   - Do the APLAI methods work for your problem?
2. There exist other constraint-based systems such as Gecode (http://www.gecode.org/) and MiniZinc (http://www.g12.csse.unimelb.edu.au/minizinc/), ...
   Select one of these alternative constraint-based systems.
   - Suppose you have to choose between the alternative system and the systems studied in this course. Give a detailed overview of the arguments on which your choice is based. Explain your final decision.
   - Write a program in the alternative system to solve Task 2.
3. Identify a related research topic.
   - Make a selection of three to five relevant research papers.
   - Write a short paper (4-5 pages) that motivates the selection of the topic, that summarizes and discusses the papers. The idea is to convince me that this topic should become a part of the APLAI course.

5 Appendix

5.1 Some useful tips

If you use LaTeX to write your report, you can use the `logicpuzzle` package for creating figures of Sudoku and Slitherlink puzzles.

Some info on how to use this package can be found here:
`http://logicpuzzle.square7.de/slitherlink`.
This is the LaTeX code for the solved puzzle in Figure 1.

```
 \begin{slitherlink}[rows=5,columns=5]
    \slitherlinkcell{2}{5}{1}
    \slitherlinkcell{1}{4}{3}
    \slitherlinkcell{5}{4}{2}
    \slitherlinkcell{1}{3}{3}
    \slitherlinkcell{2}{3}{0}
    \slitherlinkcell{3}{3}{2}
    \slitherlinkcell{5}{3}{1}
    \slitherlinkcell{1}{2}{3}
    \slitherlinkcell{2}{2}{2}
    \slitherlinkcell{3}{2}{2}
    \slitherlinkcell{3}{1}{2}
    \framearea{black}{\tikzpath{1}{6}
    {6,6,6,2,4,2,6,6,8,8,6,2,2,2,2,2,2,4,4,8,
          6,8,4,4,2,2,4,4,8,6,8,4,8,6,8,4,8}}
\end{slitherlink}
```

The `tikzpath` command takes a starting coordinate and a list of moves (where the numbers correspond to the directions on a numeric keypad). Note that the row and column numbers are swapped, and the row numbers are reversed (row 1 is at the bottom).

5.2 Some useful code

This section contains some Prolog code (CHR, Eclipse) which can help you in programming your solution.

5.2.1 Convert edge coordinate to edge index (and back)

Horizontal edges are represented as `h(Row,Col)` and vertical edges as `v(Row,Col)`.

```
edge_index(h(R,C),_,N,Index) :-
    var(Index), !, Index is (R-1)*N + (C-1).
edge_index(v(R,C),M,N,Index) :-
    var(Index), !, Index is (R-1)*(N+1) + (C-1) + (M+1)*N.
edge_index(Edge,M,N,Index) :-
    nonvar(Index),
    ( Index >= (M+1)*N ->
        Index2 is Index - (M+1)*N,
        Col is Index2 mod (N+1) + 1,
        Row is Index2 div (N+1) + 1,
        Edge = v(Row,Col)
    ;
        Col is Index mod N + 1,
        Row is Index div N + 1,
        Edge = h(Row,Col)
    ).
```

5.2.2 Printing a solution

This code should work out-of-the-box in CHR. In Eclipse, you need to add

```
:- use_module(library(listut)).
```

to import the predicate `nth0/3`.

```
print_solution(Cells,M,N,Solution) :-
    print_row(1,M,N,Solution,Cells).

print_hvalue(X) :- var(X), write('+   '), !.
print_hvalue(1) :- write('+'), !.
print_hvalue(0) :- write('+ x '), !.

print_vvalue(X) :- var(X), write('    '), !.
print_vvalue(1) :- write('|   '), !.
print_vvalue(0) :- write('    '), !.

print_vvaluem(X) :- var(X), write(' '), !.
print_vvaluem(1) :- write('|'), !.
print_vvaluem(0) :- write('x'), !.

print_cell(R,C,Cells) :-
    ( member(cell(R,C,V),Cells) ->
        write(' '), write(V), write(' ')
    ;
        write('   ')
    ).

print_row(R,M,N,Values,Cells) :-
    N2 is N + 1,
    forall( ( between(1,N ,C),
              edge_index(h(R,C),M,N,Index),
              nth0(Index,Values,Value) ),
             ( print_hvalue(Value) ) ), write('+'), nl,
    forall( ( between(1,N2,C),
              edge_index(v(R,C),M,N,Index),
              nth0(Index,Values,Value) ),
             ( print_vvalue(Value) ) ), nl,
    forall( ( between(1,N2,C),
              edge_index(v(R,C),M,N,Index),
              nth0(Index,Values,Value) ),
             ( print_vvaluem(Value), print_cell(R,C,Cells) ) ), nl,
    forall( ( between(1,N2,C),
              edge_index(v(R,C),M,N,Index),
              nth0(Index,Values,Value) ),
             ( print_vvalue(Value) ) ), nl,
    R2 is R + 1,
    ( R2 =< M + 1 ->
        print_row(R2,M,N,Values,Cells)
    ;
        true
    ).
```