

3 times Sudoku

Logic and Constraints in Clojure

Burkhardt Renz

Institut für SoftwareArchitektur
Technische Hochschule Mittelhessen

:clojureD
January 24th, 2015

The Nature of the Game

Rules and Terminology

The rule:

Each digit appears once in each unit

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

column

block

row

The Nature of the Game

Candidates for the free cells

⁵ ₉	2	4	¹ ₅	¹ _{5 6}	¹ ₆	3	8	⁵ ₇
6	³	7	2	⁵ ₈	9	1	⁵	4
8	1	⁵	7	⁴ ₅	3	²	9	6
^{1 2 3} ₅	4	8	¹ ₅ ³	^{1 2 3} _{5 6}	^{1 2} ₆	9	7	^{2 3}
^{1 2 3} ₅	³ ₇	^{2 3} ₅	¹ ₄ ³ _{5 8}	^{1 2 3} _{4 5 6 7 8 9}	^{1 2} _{4 7} ⁶	⁴ _{2 6}	² ₆	^{2 3} ₈
^{2 3}	6	9	⁴ ₈ ³	^{2 3} _{4 7 8}	² _{4 7}	5	1	^{2 3} ₈
7	5	^{2 3}	9	^{2 3}	8	² ₆	4	1
4	³ ₈	1	6	^{2 3}	5	7	²	9
²	9	6	¹ ₄	^{1 2} _{4 7}	^{1 2} _{4 7}	8	3	² ₅

The Nature of the Game

The Solution

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

The Nature of the Game

Solving Sudoku (more or less brute force)

Demo

Project <https://github.com/esb-dev/sudoku>

File `sudoku.clj`

The Nature of the Game

What, not how

- This program implements a *certain strategy* to solve the puzzle
- It says **How to solve** the puzzle
- But we want to do better – we just want to express the problem, i.e., the rules of Sudoku together with the given clues
- **What, not how** – and let a generic engine, a solver figure out *how* to solve the problem
- 3 times Sudoku = you will see 3 possibilities to express the rules of Sudoku. Each time a solver processes this specification and gives us the solution

Logic WorkBench (lwb)

About lwb

- lwb comprises functions and tools for the propositional and predicate logic
- it's a playground for a course in logic and formal methods
- work in progress, we published just some functions for the propositional logic
- <https://github.com/esb-dev/lwb>

Logic WorkBench (lwb)

Representation of propositions in lwb

- An **atomic proposition** (short: an **atom**) can have one of the truth values true or false.

It is represented in lwb as a Clojure symbol, e.g. `p`, `q`...

- A **proposition** is build from atoms and logical operators like `not`, `and`, `or`, `impl`, `ite`...

It is represented in lwb as a Clojure list – also Clojure code that way, e.g., `(or (not p) (not q))`.

Logic WorkBench (lwb)

Interesting questions in propositional logic

Given a proposition like $(\text{or } (\text{not } p) (\text{not } q))$ there are two types of questions:

- 1 Given a valuation for the atoms, i.e. an assignment of truth values for the atoms:

What is the truth value of the proposition?


Example: Given p true, q false, the truth value of the proposition above is true.


- 2 Given a proposition:
Is it **satisfiable**?, i.e. is there a valuation (a “world”) such that the proposition is true

The proposition $(\text{or } (\text{not } p) (\text{not } q))$ is satisfiable.

Logic WorkBench (lwb)

Encoding Sudoku as a proposition

~~c281?~~ ... c285?  ... ~~c289?~~

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

Logic WorkBench (lwb)

Encoding Sudoku as a proposition

- For each cell and each possible digit we introduce an atom $cxyd$, where x is the row, y the column, and d the digit in the cell.
- For all x and y exactly one of the $cxyd$ is true.
- For all units and all d at most one $cxyd$ is true
- Finally the givens, e.g., $c122$ in our example is true

Logic WorkBench (lwb)

Cardinality constraints in propositional logic

Given atoms p , q , r , we want to express:

- 1 at least one of the atoms is true
 $(\text{or } p \ q \ r)$
- 2 at most one of the atoms is true
 $(\text{or } (\text{not } p) \ (\text{not } q))$
 $(\text{or } (\text{not } p) \ (\text{not } r))$
 $(\text{or } (\text{not } q) \ (\text{not } r))$
- 3 lwb has functions that generate such formulae:
 $(\text{min-kof } k \ \text{syms})$
 $(\text{max-kof } k \ \text{syms})$
 $(\text{oneof } \text{syms})$
e.g. $(\text{oneof } '[p \ q \ r])$ gives a seq of the clauses above

Logic WorkBench (lwb)

Solving Sudoku with lwb

Demo

Project <https://github.com/esb-dev/lwb>

File `lwb/prop/examples/sudoku.clj`

Logic WorkBench (lwb)

How does it work?

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



proposition
in cnf



proposition
in dimacs



SAT4J

www.sat4j.org

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



satisfying
valuation



Kodkod in Clojure (kic)

About Kodkod and kic



- *Leopardus guigna*, Chilean cat, known to be fast
- Efficient constraint solver for first order, relational logic – a *finite model finder*
- developed by Emina Torlak, former MIT, now University of Washington, Seattle
- see <http://alloy.mit.edu/kodkod/>
- Kodkod in Clojure is an ultrathin wrapper for Kodkod, see <https://github.com/esb-dev/kic>

Kodkod in Clojure (kic)

Ingredients of a Kodkod specification

- A finite **universe** of “things”
e.g. [1 2 3 4 5 6 7 8 9]
- A **structure** of potential worlds given by relation variables
e.g., a relation variable *grid* with 3 dimensions: *x*, *y*, *d*
- A couple of **constraints** on the relational variables
e.g. $\forall x, y \exists! d : [x, y, d] \in \textit{grid}$
- A given **partial solution** defined by lower and upper bounds for the relation variables
e.g. $[1, 2, 2] \in \textit{grid} \dots$
- A **solution**, aka instance or model, is a set of relations fulfilling the constraints on the relational variables

Kodkod in Clojure (kic)

Encoding Sudoku in Kodkod

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

grid

```
...  
[2 7 1]  
[2 8 1]  
[2 8 2]  
[2 8 3]  
[2 8 4]  
[2 8 5]  
[2 8 6]  
[2 8 7]  
[2 8 8]  
[2 8 9]  
[2 9 4]  
...
```

Kodkod in Clojure (kic)

Encoding Sudoku in Kodkod

We have to express that

- each cell has just one number in it
- The value in each cell does not occur in another cell of its row
- The value in each cell does not occur in another cell of its column
- All numbers from the universe occur in a block

Kodkod/kic has a lot of operators of the relational algebra to express such constraints

Kodkod in Clojure (kic)

Solving Sudoku with kic

Demo

Project <https://github.com/esb-dev/kic>

File `kic/examples/sudoku.clj`

Kodkod in Clojure (kic)

How does it work?

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



specification
in kodkod



proposition
in dimacs



SAT Solver

e.g. SAT4J

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



model for
the spec



satisfying
valuation



core.logic

About core.logic

- core.logic is a logic programming library for Clojure and ClojureScript
- core.logic is a port of miniKanren, developed first in Scheme by William E. Byrd, Daniel P. Friedman and others
- core.logic expands Clojure from functional to relational programming and constraint solving
- core.logic supports CLP(FD) – Constraint Logic Programming over finite domains

core.logic

Basic concepts of core.logic

- A **logic variable** aka *lvar* is a placeholder for a value
- **Goals** are functions that encapsulate a logic program
- A goal **succeeds** if it can be made true by substituting variables by values
- Goals can be combined by conjunction (*and*) and disjunction (*or*)
- If a goal succeeds, we get as a result a sequence of substitutions of the logic variables, i.e. **all possible “worlds”** that satisfy the constraints of the logic program

core.logic

Basic concepts of core.logic – Example

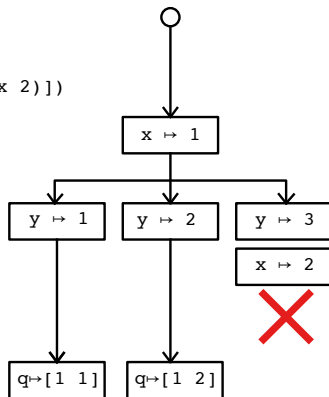
```
(run* [q]
  (fresh [x y]
    (== x 1)
    (conde [(== y 1)]
            [(== y 2)]
            [(== y 3) (== x 2)]))
  (== q [x y]))
; => ([1 1] [1 2])
```

- logic variable `q` aka query
- local fresh (unbound) logic variables `x` and `y`
- `==` means unify `x` with `1`
- multiple goals connected by *and*
- `conde` is *or* (conditional every line)
- `run*` runs the solver and gives all possible solutions

core.logic

Unification – example

```
(run* [q]
  (fresh [x y]
    (== x 1)
    (conde [(== y 1)]
            [(== y 2)]
            [(== y 3) (== x 2)]))
  (== q [x y])))
```



`;=> ([1 1] [1 2])`

core.logic

Solving Sudoku in core.logic

Demo

Project <https://github.com/esb-dev/sudoku>

File sudoku-cl.clj

Play with logic in Clojure!
It's fun – and useful, too

Appendix

Benchmarks

	easy50	top95	hardest	relative
python (P. Norvig)	4.0	14.6	5.3	1.0
brute force	53.4	5838.2	213.6	340.0
lwb	136.7	141.0	137.5	13.2
kic	14.2	13.7	10.9	1.3
core.logic	22.2	5317.3 ¹	258.13	308.9

(times in msec)

¹A strange result. There are some puzzles in top95.txt where core.logic needs 25 secs, others are solved in about 1 sec. I don't know the reason for such a huge difference.

Appendix

Interesting Facts about Sudoku

- 1 The number of complete Sudoku grids is

$$9! \times 722 \times 27 \times 27,704,267,971 \approx 6.7 \times 10^{21}$$

(Bertram Felgenhauer and Frazer Jarvis, 2005)

- 2 The minimal number of givens necessary to build proper puzzles is 17.

(Gary McGuire, Bastian Tugemann and Gilles Civario, 2012)

- 3 The general problem of solving a Sudoku for order n has been shown to be NP-complete.

(Takayugi Yato, 2003)

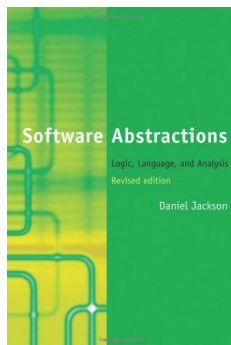
Appendix

Usages of SAT solvers

- static code analysis
- variability model of (software) product lines
- analysis of component-based systems
- analysis of genetic networks in bioinformatics
- ...

Appendix

Usages of Kodkod



- **Alloy** a lightweight formal method is build upon Kodkod
- Analysis of specifications, code, designs . . .
- Alloy finds small sized models or counterexamples for a given Alloy specification
- *small scope hypothesis*
- awesome to develop Alloy specs interactively in the **Alloy Analyzer**
- Introduction to Alloy and the Alloy Analyzer (in german) on my web site: <https://homepages.thm.de/~hg11260/lfm.html>

Appendix

Usages of core.logic

- **kibit**, a static code analyzer for Clojure
- **damp.ekeko**, an Eclipse plugin for inspection and manipulation of files in a workspace
- **Funny QT**, a model querying and transformation library
- **natural-deduction**, a proof system for the propositional and first order logic, developed at the THM
<https://github.com/Kuerschten/natural-deduction>

core.logic – How to get started



David Nolen et al.

A Core.logic Primer

<https://github.com/clojure/core.logic/wiki/A-Core.logic-Primer>



Russell Mull

microLogic

<http://mullr.github.io/micrologic/literate.html>



Bruce A. Tate et al.

Seven More Languages in Seven Weeks, Chap. 6

The Pragmatic BookShelf, 2014