Formalising and Proving with Sudokus

Jonas Betzendahl
jonas.betzendahl@fau.de

WuV-Seminar
2021 – 06 – 24

Motivation

My current research question are *generated* theorem provers for MMT theories.

The idea being that a user could formalise whatever theory they needed, could then press a button and get a program that can solve some proof obligations for them, using the rules of their new theory.

Nobody expects these provers to produce mathematically interesting insights or prove big theorems, but if we can take the busywork off the user's hands, that's already very helpful.

Test Cases

I anticipate the most use for these provers for simple proof obligations to be in the field of Undefinedness and Soft Types, two other topics of my PhD program.

Both of these have a tendency to generate loads of tiny obligations that would be prohibitively tedious to prove yourself all the time. Tests in the realm of Propositional Logic have been successfull.

Test Cases

I anticipate the most use for these provers for simple proof obligations to be in the field of Undefinedness and Soft Types, two other topics of my PhD program.

Both of these have a tendency to generate loads of tiny obligations that would be prohibitively tedious to prove yourself all the time. Tests in the realm of Propositional Logic have been successfull.

Hence, I thought that solving Sudokus (well-known, easy to conceptualise and famously NP-complete [in the generalised case]) might make a good case study. This talk is about what I found out.

Sudoku

The game, the strategies and my formalisation.

The Game of Sudoku

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

Sudoku (jpn.: "Single Number") is a popular logic-based, combinatorial number puzzle.

- 81 *cells* arranged in (partially filled) square grid.
- Rows, Colums and 3x3 Boxes form connected *houses*.
- In a solution, each house needs to contain each digit $(1 - 9)$ exactly once.

Pencil Marks

A common approach for both humans and computers trying to solve Sudokus is to keep track of which digits could possibly or need to be necessarily filled into which cells with so-called *Pencil Marks*.

Formalisation (1): Kripke Models

The *possibility* and *necessity* of digits being filled into cells being such a central element to the puzzle makes applying *Modal Logic* an obvious choice. We will be following Daan Di Scala's formalisation of Sudoku as a Kripke Model[1].

Definition 2.2.1 (Sudoku Kripke Model).
Our Sudoku model is a Kripke model $S = \langle W, R, D \rangle$ such that

- $W = C \cup P = \{c_1, ..., c_{81}\} \cup \{p_1, ..., p_9\}$, a union of C, the set of worlds representing Sudoku cells, and P, the set of possible pencil marks.
- $R = G \cup P = H \cup V \cup B \cup P$, a union of sets of different accessibility relations between these worlds, as defined in Definition 2.2.2.
- $D = \{1, ..., 9\}$ being a set of Digit valuations.
  Proposition d is true in world p if $d \in D(p)$, and false in p if $d \notin D(p)$.

The set of accessibility relations of this Kripke Model, R, is made up of four different sets of accessibility relations, which are called H, V, B, and P.

[1] Available at: `http://dspace.library.uu.nl/handle/1874/396282`

Formalisation (2): Modal Logic

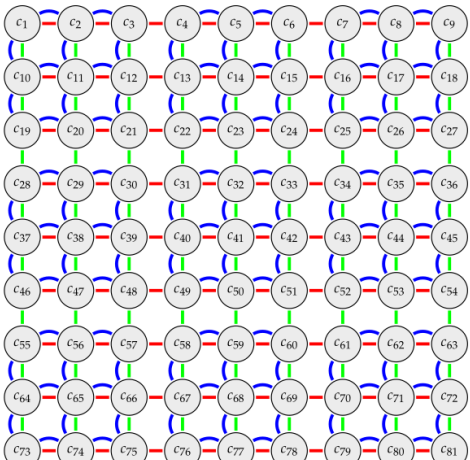
In this framework, we have $81 + 9 = 90$ worlds and "Solving the Sudoku" means reasoning about the accessibility relations until each cell-world has access to exactly one digit-world.

In the following, these notations will be used:

$c_x \; B \; c_y$	c_x and c_y are in the same block
$S, c_x \vDash \Box_V \varphi$	φ is the case in all cells in the same column.
$S, c_{21} \vDash \Box_p(3 \otimes 5)$	Cell c_{21} necessarily contains either a 3/5.
$S, c_{81} \vDash \Box_H(\diamond_p 6 \wedge \diamond_p 8)$	Cells hor. reachable from cell c_{81} possibly contain a 6/8.

Reasoning will happen inside a ND calculus.

Formalisation (3): House Relations



There are three house relations:

- Same Row
- Same Column
- Same Box

These are symmetric and transitive but (at least in this writeup) not reflexive.

They are realised through (an avalanche of) single constants stating the connection as axioms.

Formalisation (4): Helpers

The formalisation of the Sudoku puzzle itself is relatively straightforward. However, to reasonably talk about the rules of the game, it is often necessary to implement helpers like this one.
As you can see, these can become quite unwieldy.

```
all_different : { T, a : tm T, b : tm T, c : tm T, d : tm T, e : tm T, f : tm T, g : tm T, h : tm T, i : tm T } prop
              = [T,p1,p2,p3,p4,p5,p6,p7,p8,p9]
                ((p1 ≠ p2) ∧ (p1 ≠ p3) ∧ (p1 ≠ p4) ∧ (p1 ≠ p5) ∧ (p1 ≠ p6) ∧ (p1 ≠ p7) ∧ (p1 ≠ p8) ∧ (p1 ≠ p9) ∧ (p2 ≠ p3)
              ∧ (p2 ≠ p4) ∧ (p2 ≠ p5) ∧ (p2 ≠ p6) ∧ (p2 ≠ p7) ∧ (p2 ≠ p8) ∧ (p2 ≠ p9) ∧ (p3 ≠ p4) ∧ (p3 ≠ p5) ∧ (p3 ≠ p6)
              ∧ (p3 ≠ p7) ∧ (p3 ≠ p8) ∧ (p3 ≠ p9) ∧ (p4 ≠ p5) ∧ (p4 ≠ p6) ∧ (p4 ≠ p7) ∧ (p4 ≠ p8) ∧ (p4 ≠ p9) ∧ (p5 ≠ p6)
              ∧ (p5 ≠ p7) ∧ (p5 ≠ p8) ∧ (p5 ≠ p9) ∧ (p6 ≠ p7) ∧ (p6 ≠ p8) ∧ (p6 ≠ p9) ∧ (p7 ≠ p8) ∧ (p7 ≠ p9) ∧ (p8 ≠ p9))
              |# ≺ 2 3 4 5 6 7 8 9 10 ≻|

swap12 : { T, d1 : tm T, d2 : tm T, d3 : tm T, d4 : tm T, d5 : tm T, d6 : tm T, d7 : tm T, d8 : tm T, d9 : tm T }
       ⊢ (≺ d1 d2 d3 d4 d5 d6 d7 d8 d9 ≻) → ⊢ (≺ d2 d1 d3 d4 d5 d6 d7 d8 d9 ≻)|
```

Strategy 1: Cell-Based Elimination

"If a cell already contains a certain digit, it cannot contain another digit"

$$S, c \vDash \Box_P d_1$$

$$\ldots$$

$$S, c \vDash \neg\Diamond_P d_2$$

Formalising Strategy 1

```
// Cell-Based Elimination
theory SudokuStrategyCBE =
    include ?Sudoku
    cbe : {d1 : tm PM, d2 : tm PM, c : tm cell} (⊢ (d1 ≠ d2)) → (⊢ (□ (c ∏ d1))) → (⊢ (¬ (◇ (c ∏ d2))))
```

Formalising Strategy 1

```
// Cell-Based Elimination
theory SudokuStrategyCBE =
    include ?Sudoku
    cbe : {d1 : tm PM, d2 : tm PM, c : tm cell} (⊢ (d1 ≠ d2)) → (⊢ (□ (c ⊓ d1))) → (⊢ (¬ (◇ (c ⊓ d2))))
```

Straightforward and easy to write down. :)

Strategy 2: Pencil-Mark Duality

"[W]ith a total of n digits, if in a certain cell there are m PMs possible and $n - m$ PMs not possible, this means that the digit to fill in must be [one] of the m Pencil Marks"

$$S, c \vDash \Diamond_P d_1$$
$$S, c \vDash \neg\Diamond_P d_2 \wedge \ldots \wedge \neg\Diamond_P d_9$$
$$\ldots$$
$$S, c \vDash \Box_P d_1$$

(A) PMD1

$$S, c \vDash \Diamond_P d_1 \wedge \Diamond_P d_2$$
$$S, c \vDash \neg\Diamond_P d_3 \wedge \ldots \wedge \neg\Diamond_P d_9$$
$$\ldots$$
$$S, c \vDash \Box_P (d_1 \otimes d_2)$$

(B) PMD2

Formalising Strategy 2

```
pmd1 : { d1 : tm PM, d2 : tm PM, d3 : tm PM, d4 : tm PM, d5 : tm PM, d6 : tm PM, d7 : tm PM, d8 : tm PM, d9 : tm PM , c : tm cell }
    (⊢ (≪ d1 d2 d3 d4 d5 d6 d7 d8 d9 ≫))
    → (⊢ (◇ (c Π d1)))
    → (⊢ (¬ (◇ (c Π d2)))) → (⊢ (¬ (◇ (c Π d3)))) → (⊢ (¬ (◇ (c Π d4)))) → (⊢ (¬ (◇ (c Π d5))))
    → (⊢ (¬ (◇ (c Π d6)))) → (⊢ (¬ (◇ (c Π d7)))) → (⊢ (¬ (◇ (c Π d8)))) → (⊢ (¬ (◇ (c Π d9))))
    → (⊢ (□ (c Π d1)))
```

Formalising Strategy 2

```
pmd1 : { d1 : tm PM, d2 : tm PM, d3 : tm PM, d4 : tm PM, d5 : tm PM, d6 : tm PM, d7 : tm PM, d8 : tm PM, d9 : tm PM , c : tm cell }
     (⊢ (≪ d1 d2 d3 d4 d5 d6 d7 d8 d9 ≫))
     → (⊢ (◇ (c Π d1)))
     → (⊢ (¬ (◇ (c Π d2)))) → (⊢ (¬ (◇ (c Π d3)))) → (⊢ (¬ (◇ (c Π d4)))) → (⊢ (¬ (◇ (c Π d5))))
     → (⊢ (¬ (◇ (c Π d6)))) → (⊢ (¬ (◇ (c Π d7)))) → (⊢ (¬ (◇ (c Π d8)))) → (⊢ (¬ (◇ (c Π d9))))
     → (⊢ (□ (c Π d1)))
```

This one is a little more difficult.
Also, this is showing only one of many instances of this rule.

Strategy 3: House-Based Elimination

"If a cell in a house already contains a digit, another cell in the same house cannot contain the same digit"

$$S, c_1 \models \Box_p d$$

$$c_1 G c_2$$

$$\ldots$$

$$S, c_2 \models \neg \Diamond_p d$$

Formalising Strategy 3

```
theory SudokuStrategyHBE =
    include ?Sudoku
    hbe_hori : { c1 : tm cell, c2 : tm cell, d : tm PM } (⊢ (□ (c1 ⊓ d))) → (⊢ (c1 𝒩 c2)) → (⊢ (¬ (◇ (c2 ⊓ d))))
    hbe_vert : { c1 : tm cell, c2 : tm cell, d : tm PM } (⊢ (□ (c1 ⊓ d))) → (⊢ (c1 𝒱 c2)) → (⊢ (¬ (◇ (c2 ⊓ d))))
    hbe_blck : { c1 : tm cell, c2 : tm cell, d : tm PM } (⊢ (□ (c1 ⊓ d))) → (⊢ (c1 ℬ c2)) → (⊢ (¬ (◇ (c2 ⊓ d))))
```

Formalising Strategy 3

```
theory SudokuStrategyHBE =
    include ?Sudoku
    hbe_hori : { c1 : tm cell, c2 : tm cell, d : tm PM } (⊢ (□ (c1 ⊓ d))) → (⊢ (c1 𝒽 c2)) → (⊢ (¬ (◇ (c2 ⊓ d))))
    hbe_vert : { c1 : tm cell, c2 : tm cell, d : tm PM } (⊢ (□ (c1 ⊓ d))) → (⊢ (c1 𝒱 c2)) → (⊢ (¬ (◇ (c2 ⊓ d))))
    hbe_blck : { c1 : tm cell, c2 : tm cell, d : tm PM } (⊢ (□ (c1 ⊓ d))) → (⊢ (c1 ℬ c2)) → (⊢ (¬ (◇ (c2 ⊓ d))))
```

Multiple rules needed, but not too bad.
This would be fine if all rules were like this.

Strategy 4: Last Remaining Cell

"If all the other cells in the same house cannot possibly contain a certain digit, then this cell must contain that digit"

$$S, c \models \Box_G \neg \Diamond_P d$$

$$S, c \models \Diamond_P d$$

$$\ldots$$

$$S, c \models \Box_P d$$

Formalising Strategy 4

```
hH : { c : tm cell, k : tm cell, p : tm PM } prop| = [c1,k1,p1] (c1 ≠ k1) ∧ (c1 𝒦 k1) ∧ (¬ (◇ (k1 Π p1))) |# ℍ 1 2 3|
hV : { c : tm cell, k : tm cell, p : tm PM } prop| = [c1,k1,p1] (c1 ≠ k1) ∧ (c1 𝒱 k1) ∧ (¬ (◇ (k1 Π p1))) |# 𝕍 1 2 3|
hB : { c : tm cell, k : tm cell, p : tm PM } prop| = [c1,k1,p1] (c1 ≠ k1) ∧ (c1 𝒷 k1) ∧ (¬ (◇ (k1 Π p1))) |# 𝔹 1 2 3|

lrc_hori : { c1 : tm cell, c2 : tm cell, c3 : tm cell, c4 : tm cell, c5 : tm cell, c6 : tm cell, c7 : tm cell, c8 : tm cell, c9 : tm cell, p : tm PM }
         (⊢ (≺ c1 c2 c3 c4 c5 c6 c7 c8 c9 ≻))
         → ⊢ ℍ c1 c2 p → ⊢ ℍ c1 c3 p → ⊢ ℍ c1 c4 p → ⊢ ℍ c1 c5 p → ⊢ ℍ c1 c6 p → ⊢ ℍ c1 c7 p → ⊢ ℍ c1 c8 p → ⊢ ℍ c1 c9 p
         → ⊢ (□ (c1 Π p))|
```

Formalising Strategy 4

```
hH : { c : tm cell, k : tm cell, p : tm PM } prop| = [c1,k1,p1] (c1 ≠ k1) ∧ (c1 𝒦 k1) ∧ (¬ (◇ (k1 Π p1))) |# ℍ 1 2 3|
hV : { c : tm cell, k : tm cell, p : tm PM } prop| = [c1,k1,p1] (c1 ≠ k1) ∧ (c1 𝒱 k1) ∧ (¬ (◇ (k1 Π p1))) |# 𝕍 1 2 3|
hB : { c : tm cell, k : tm cell, p : tm PM } prop| = [c1,k1,p1] (c1 ≠ k1) ∧ (c1 𝒷 k1) ∧ (¬ (◇ (k1 Π p1))) |# 𝔹 1 2 3|

lrc_hori : { c1 : tm cell, c2 : tm cell, c3 : tm cell, c4 : tm cell, c5 : tm cell, c6 : tm cell, c7 : tm cell, c8 : tm cell, c9 : tm cell, p : tm PM }
         (⊢ (≺ c1 c2 c3 c4 c5 c6 c7 c8 c9 ≻))
         → ⊢ ℍ c1 c2 p → ⊢ ℍ c1 c3 p → ⊢ ℍ c1 c4 p → ⊢ ℍ c1 c5 p → ⊢ ℍ c1 c6 p → ⊢ ℍ c1 c7 p → ⊢ ℍ c1 c8 p → ⊢ ℍ c1 c9 p
         → ⊢ (□ (c1 Π p))|
```

Pretty bad. Three helper constants and still the three rules (only one shown) are getting quite convoluted.

Strategy 5: Pencil Mark Introduction

"If in none of the three houses of a cell another cell must contain a digit, this cell can possibly contain this digit"

$$
\begin{array}{l}
S, c \vDash \Box_H \neg \Box_P d \\
S, c \vDash \Box_V \neg \Box_P d \\
S, c \vDash \Box_B \neg \Box_P d \\
\ldots \\
S, c \vDash \Diamond_P d
\end{array}
$$

Formalising Strategy 5

```
allB : { c1 : tm cell, c2 : tm cell, c3 : tm cell, c4 : tm cell, c5 : tm cell, c6 : tm cell, c7 : tm cell, c8 : tm cell, c9 : tm cell } prop
     = [c1,c2,c3,c4,c5,c6,c7,c8,c9] (c1 ⅋ c2) ∧ (c2 ⅋ c3) ∧ (c3 ⅋ c4) ∧ (c4 ⅋ c5) ∧ (c5 ⅋ c6) ∧ (c6 ⅋ c7) ∧ (c7 ⅋ c8) ∧ (c8 ⅋ c9)

house : { c1 : tm cell, c2 : tm cell, c3 : tm cell, c4 : tm cell, c5 : tm cell, c6 : tm cell, c7 : tm cell, c8 : tm cell, c9 : tm cell, p : tm PM } prop
      = [c1,c2,c3,c4,c5,c6,c7,c8,c9,p] (¬(□(c2 ⊓ p))) ∧ (¬(□(c3 ⊓ p))) ∧ (¬(□(c4 ⊓ p)))
      ∧ (¬(□(c5 ⊓ p))) ∧ (¬(□(c6 ⊓ p))) ∧ (¬(□(c7 ⊓ p))) ∧ (¬(□(c8 ⊓ p))) ∧ (¬(□(c9 ⊓ p)))

pmi : { pivot : tm cell, p : tm PM,
        c12 : tm cell, c13 : tm cell, c14 : tm cell, c15 : tm cell, c16 : tm cell, c17 : tm cell, c18 : tm cell, c19 : tm cell,
        c22 : tm cell, c23 : tm cell, c24 : tm cell, c25 : tm cell, c26 : tm cell, c27 : tm cell, c28 : tm cell, c29 : tm cell,
        c32 : tm cell, c33 : tm cell, c34 : tm cell, c35 : tm cell, c36 : tm cell, c37 : tm cell, c38 : tm cell, c39 : tm cell }
      (⊢ (≪ pivot c12 c13 c14 c15 c16 c17 c18 c19 ≫)) → ⊢ (allV pivot c12 c13 c14 c15 c16 c17 c18 c19) → ⊢ (house pivot c12 c13 c14 c15 c16 c17 c18 c19 p)
    → (⊢ (≪ pivot c22 c23 c24 c25 c26 c27 c28 c29 ≫)) → ⊢ (allH pivot c12 c13 c14 c15 c16 c17 c18 c19) → ⊢ (house pivot c22 c23 c24 c25 c26 c27 c28 c29 p)
    → (⊢ (≪ pivot c32 c33 c34 c35 c36 c37 c38 c39 ≫)) → ⊢ (allB pivot c12 c13 c14 c15 c16 c17 c18 c19) → ⊢ (house pivot c32 c33 c34 c35 c36 c37 c38 c39 p)
    → ⊢ (◇ (pivot ⊓ p))
```

Formalising Strategy 5

```
allB : { c1 : tm cell, c2 : tm cell, c3 : tm cell, c4 : tm cell, c5 : tm cell, c6 : tm cell, c7 : tm cell, c8 : tm cell, c9 : tm cell } prop
     = [c1,c2,c3,c4,c5,c6,c7,c8,c9] (c1 𝒷 c2) ∧ (c2 𝒷 c3) ∧ (c3 𝒷 c4) ∧ (c4 𝒷 c5) ∧ (c5 𝒷 c6) ∧ (c6 𝒷 c7) ∧ (c7 𝒷 c8) ∧ (c8 𝒷 c9)

house : { c1 : tm cell, c2 : tm cell, c3 : tm cell, c4 : tm cell, c5 : tm cell, c6 : tm cell, c7 : tm cell, c8 : tm cell, c9 : tm cell, p : tm PM } prop
      = [c1,c2,c3,c4,c5,c6,c7,c8,c9,p] (¬(□(c2 ⊓ p))) ∧ (¬(□(c3 ⊓ p))) ∧ (¬(□(c4 ⊓ p)))
      ∧ (¬(□(c5 ⊓ p))) ∧ (¬(□(c6 ⊓ p))) ∧ (¬(□(c7 ⊓ p))) ∧ (¬(□(c8 ⊓ p))) ∧ (¬(□(c9 ⊓ p)))

pmi : { pivot : tm cell, p : tm PM,
        c12 : tm cell, c13 : tm cell, c14 : tm cell, c15 : tm cell, c16 : tm cell, c17 : tm cell, c18 : tm cell, c19 : tm cell,
        c22 : tm cell, c23 : tm cell, c24 : tm cell, c25 : tm cell, c26 : tm cell, c27 : tm cell, c28 : tm cell, c29 : tm cell,
        c32 : tm cell, c33 : tm cell, c34 : tm cell, c35 : tm cell, c36 : tm cell, c37 : tm cell, c38 : tm cell, c39 : tm cell }
      (⊢ (≪ pivot c12 c13 c14 c15 c16 c17 c18 c19 ≫)) → ⊢ (allV pivot c12 c13 c14 c15 c16 c17 c18 c19) → ⊢ (house pivot c12 c13 c14 c15 c16 c17 c18 c19 p)
    → (⊢ (≪ pivot c22 c23 c24 c25 c26 c27 c28 c29 ≫)) → ⊢ (allH pivot c12 c13 c14 c15 c16 c17 c18 c19) → ⊢ (house pivot c22 c23 c24 c25 c26 c27 c28 c29 p)
    → (⊢ (≪ pivot c32 c33 c34 c35 c36 c37 c38 c39 ≫)) → ⊢ (allB pivot c12 c13 c14 c15 c16 c17 c18 c19) → ⊢ (house pivot c32 c33 c34 c35 c36 c37 c38 c39 p)
    → ⊢ (◇ (pivot ⊓ p))
```

Multiple helpers and still the rule itself is gigantic (26 arguments!).
Absolutely terrible for what seems such a basic rule.

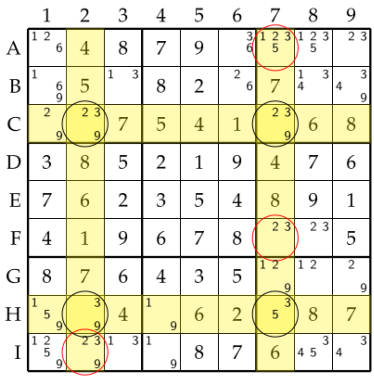
Intermediate & Advanced Strategies

There are four more strategies on the Intermediate and Advanced Level, quickly rising in complexity.

Difficulty	Strategy	Abbreviation(s)	Subsection
Implicit	Cell Based Elimination	CBE	(2.3.1)
Implicit	Pencil Mark Duality	PMD1, PMD2, PMD3	(2.3.2)
Beginner	House Based Elimination	HBE	(2.3.3)
Beginner	Last Remaining Cell	LRC	(2.3.4)
Beginner	Pencil Mark Introduction	PMI	(2.3.5)
Intermediate	Naked Tuples Elimination	N2E, N3E	(2.3.6)
Intermediate	Pointing Tuples Elimination	P2E, P3E	(2.3.7)
Advanced	Hidden Pairs Elimination	H2E	(2.3.8)
Advanced	X-Wing Elimination	XWE	(2.3.9)

TABLE 2.1: Strategies ordered by difficulty

X-Wing Elimination...



2.3.9 X-Wing Elimination (Advanced)

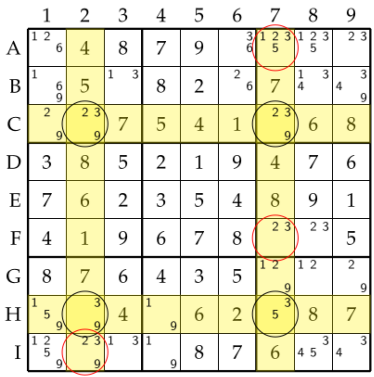
$$
\begin{array}{l}
S, c_1, c_2 \models \Diamond_P d \\
S, c_3, \ldots, c_9 \models \neg\Diamond_P d \\
c_1 H c_2 \wedge \ldots \wedge c_1 H c_9 \\
S, c_{10}, c_{11} \models \Diamond_P d \\
S, c_{12}, \ldots, c_{18} \models \neg\Diamond_P d \\
c_{10} H c_{11} \wedge \ldots \wedge c_{10} H c_{18} \\
S, c_{19} \models \Diamond_P d \\
c_1 V c_{19} \vee c_{10} V_{19} \\
\ldots \\
S, c_{19} \models \neg\Diamond_P d
\end{array}
$$

(A) Horizontal XWE

$$
\begin{array}{l}
S, c_1, c_2 \models \Diamond_P d \\
S, c_3, \ldots, c_9 \models \neg\Diamond_P d \\
c_1 V c_2 \wedge \ldots \wedge c_1 V c_9 \\
S, c_{10}, c_{11} \models \Diamond_P d \\
S, c_{12}, \ldots, c_{18} \models \neg\Diamond_P d \\
c_{10} V c_{11} \wedge \ldots \wedge c_{10} V c_{18} \\
S, c_{19} \models \Diamond_P d \\
c_1 H c_{19} \vee c_{10} H_{19} \\
\ldots \\
S, c_{19} \models \neg\Diamond_P d
\end{array}
$$

(B) Vertical XWE

X-Wing Elimination...



2.3.9 X-Wing Elimination (Advanced)

$$
\begin{array}{l}
S, c_1, c_2 \models \Diamond_P d \\
S, c_3, \ldots, c_9 \models \neg\Diamond_P d \\
c_1 H c_2 \wedge \ldots \wedge c_1 H c_9 \\
S, c_{10}, c_{11} \models \Diamond_P d \\
S, c_{12}, \ldots, c_{18} \models \neg\Diamond_P d \\
c_{10} H c_{11} \wedge \ldots \wedge c_{10} H c_{18} \\
S, c_{19} \models \Diamond_P d \\
c_1 V c_{19} \vee c_{10} V_{19} \\
\ldots \\
S, c_{19} \models \neg\Diamond_P d
\end{array}
$$

(A) Horizontal XWE

$$
\begin{array}{l}
S, c_1, c_2 \models \Diamond_P d \\
S, c_3, \ldots, c_9 \models \neg\Diamond_P d \\
c_1 V c_2 \wedge \ldots \wedge c_1 V c_9 \\
S, c_{10}, c_{11} \models \Diamond_P d \\
S, c_{12}, \ldots, c_{18} \models \neg\Diamond_P d \\
c_{10} V c_{11} \wedge \ldots \wedge c_{10} V c_{18} \\
S, c_{19} \models \Diamond_P d \\
c_1 H c_{19} \vee c_{10} H_{19} \\
\ldots \\
S, c_{19} \models \neg\Diamond_P d
\end{array}
$$

(B) Vertical XWE

. . . not considered. Too complex.

ELPI & Provers

Prolog for proving

What is ELPI?

λProlog is a programming language based on an intuitionistic fragment of Church's Simple Theory of Types that extends Prolog by features such as abstract data types and higher-order programming, adding Harrop Formulas to Prolog's underlying foundation of Horn clauses.

λProlog is interesting for our problem since we can use its higher-order unification algorithm for proof search.

What is ELPI?

λProlog is a programming language based on an intuitionistic fragment of Church's Simple Theory of Types that extends Prolog by features such as abstract data types and higher-order programming, adding Harrop Formulas to Prolog's underlying foundation of Horn clauses.

λProlog is interesting for our problem since we can use its higher-order unification algorithm for proof search.

The ELPI system is an implementation of λProlog with additional features and a focus on performance, moving some previously intractable proof searches into reach. It is implemented in the OCaml programming language.

The Generated Provers

I am picking up on previous work by Cohen, Rabe and Schaefer. For any given MMT theory, one can generate an ELPI *kernel*, a file that introduces the declarations of a theory and ELPI rules that reflect the inference rules of the theory (e.g. $\wedge_I$ or $\neg_E$).

The Generated Provers

I am picking up on previous work by Cohen, Rabe and Schaefer. For any given MMT theory, one can generate an ELPI *kernel*, a file that introduces the declarations of a theory and ELPI rules that reflect the inference rules of the theory (e.g. $\wedge_I$ or $\neg_E$).

One also generates *experts*, i.e. ELPI modules that apply strategies and/or apply help predicates to reduce the enormous search space of proof terms that would otherwise be intractable.

Proof Certificates

The behaviour and the output of these ELPI provers can be guided by the user through selection of so-called "Proof Certificates".

These can be the simple difference between a straightforward iterative deepening proof or the more sophisticated backchaining proof where certain rules are only applied under certain circumstances (to manage search space).

[2] See: `http://www.lix.polytechnique.fr/Labo/Dale.Miller/ProofCert/`)

Proof Certificates

The behaviour and the output of these ELPI provers can be guided by the user through selection of so-called "Proof Certificates".

These can be the simple difference between a straightforward iterative deepening proof or the more sophisticated backchaining proof where certain rules are only applied under certain circumstances (to manage search space).

They can also deliver an actual proof certificates (compare Dale Miller's ProofCert project[2] which contain a witness of the successful proof. These can become extremely important as they might need to be checked by MMT at some point (because relying on ELPI for proofs can introduce trust issues).

[2] See: `http://www.lix.polytechnique.fr/Labo/Dale.Miller/ProofCert/`)

Example Provers (1)

Here's a snippet of a generated prover for a rule for negation elimination:

```
/* generated from notElimination : {A}⊢¬¬A⟶⊢A */

ded X2 A :- help/notElimination X2 A X1 , ded X1 (not (not A)).

help/notElimination (prodcert X3/1 X3/2) A (prodcert X2/1 X2/2) :- help/notElimination X3/1 A X2/1 , help/notElimination X3/2 A X2/2.

help/notElimination (idcert X3) A (idcert X2) :- X3 > 0 , X2 is X3 - 1.

help/notElimination (ptcert (notElimination A X2)) A (ptcert X2).

help/notElimination (ucert X2) A (ucert (prep (ded X3 (not (not A))) X2)) :- ded X3 (not (not A)) , occatmost (ded X3 (not (not A))) 1 X2.

help/notElimination (hdcert X2) A (hdcert X2).

help/notElimination (hd2cert X2) A (hd2cert X2).

help/notElimination (bccert X3) A (bccert X2) :- bc/pos X3 , X2 is X3 - 1.

```

Example Provers (2)

This is how a test-case looks like:

```
accumulate "generic/bc".
accumulate "generated/plnd".
accumulate "generic/idprove".

id_prove_run N Res F :- ded (prodcert (bccert N) (ptcert Res)) F.

mytest P :- pi a \ pi b \ P (impl a (impl b a)).
mytest P :- pi a \ pi b \ P (impl (and a b) a).
mytest P :- pi a \ pi b \ P (impl (or a b) (or b a)).
mytest P :- pi a \ pi b \ P (impl (and a b) (impl (impl a (impl b false)) false)).
mytest P :- pi a \ pi b \ pi c \ P (impl (impl a (or b c)) (impl (impl b c) (impl a c))).
mytest P :- pi a \ pi b \ P (or a (not a)).
mytest P :- pi a \ pi b \ P (or (not a) a).

main :- mytest id_prove.
main.
```

live demo...

Results

Does it work as described for Sudokus?

No.

What works?

Formalisation of Implicit and Beginner Strategies (Di Scala's classification) is complete.

What works?

Formalisation of Implicit and Beginner Strategies (Di Scala's classification) is complete.

Currently, the generated prover can prove simple facts about a given Sudoku (e.g. $c_i \mathcal{V} c_j$ given that $c_j \mathcal{V} c_i$ and the fact that $\mathcal{V}$ is symmetric).

What works?

Formalisation of Implicit and Beginner Strategies (Di Scala's classification) is complete.

Currently, the generated prover can prove simple facts about a given Sudoku (e.g. $c_i \mathcal{V} c_j$ given that $c_j \mathcal{V} c_i$ and the fact that $\mathcal{V}$ is symmetric).

The prover seems "stuck" after only a few layers of iterative deepening, which Frederick and I determined not to be an actual infinite loop, but simply a combinatorial explosion.

What does not?

Formalisation of Intermediate and Advanced Strategies has not started (no need to waste time on that).

What does not?

Formalisation of Intermediate and Advanced Strategies has not started (no need to waste time on that).

Even the simplest steps (solving a Sudoku from a completely filled board sans one cell) is currently too complicated for generated provers.

Major Problems?

- **MMT is good for the logic-based part but bad for the combinatorial part**
  Some preconditions are incredibly complicated to write and require a lot of proving power
- **The formalisation of modal logic is incomplete**
  LATIN2 contains the basics but needs more rules and theorems that the provers can use.
- **Backchaining Algorithm is not up to par for this task**
  There needs to be a better way of reducing search space.

Outlook

Where do we go from here?

Plans

In the short term, I will...

- ...improve the Backchaining algorithm.
- ...see if the provers are more helpful with more hand-encoded domain knowledge.
- ...focus on test cases with less combinatorial explosion.

Plans

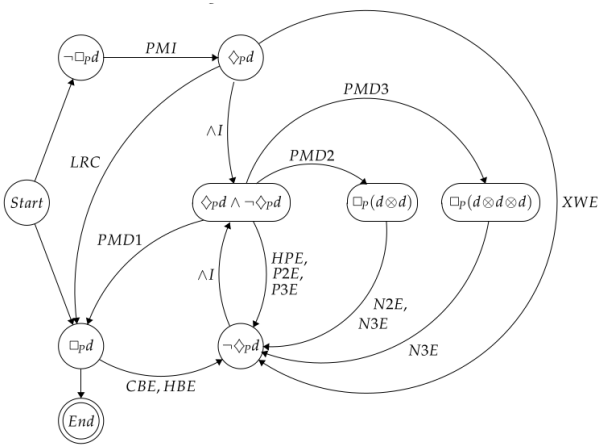
In the short term, I will...

- ...improve the Backchaining algorithm.
- ...see if the provers are more helpful with more hand-encoded domain knowledge.
- ...focus on test cases with less combinatorial explosion.

In the long term, I would like to...

- ...see if I can re-work my formalisation.
- ...try to find more ways to easily encode domain knowledge that can be used by provers.
- ...revisit Sudokus as a hard test case.

Strategy Graph



We could use the so-called "Strategy Graph" from Di Scala's thesis as a way to reduce the search space.

This could also function as a case study for encoding domain knowledge as effectively usable for the provers.

Conclusion

We set out to investigate whether the problem of Sudoku can be solved with the current state of the generated ELPI provers.

Conclusion

We set out to investigate whether the problem of Sudoku can be solved with the current state of the generated ELPI provers.

I still believe Sudoku is an interesting case study for this problem space!

Conclusion

We set out to investigate whether the problem of Sudoku can be solved with the current state of the generated ELPI provers.

I still believe Sudoku is an interesting case study for this problem space!
. . . but a lot needs to happen before it becomes feasible.

Conclusion

We set out to investigate whether the problem of Sudoku can be solved with the current state of the generated ELPI provers.

I still believe Sudoku is an interesting case study for this problem space!
. . . but a lot needs to happen before it becomes feasible.

This experiment has, however clearly demonstrated where the current state's weaknesses are and what needs to be tackled next.

Conclusion

We set out to investigate whether the problem of Sudoku can be solved with the current state of the generated ELPI provers.

I still believe Sudoku is an interesting case study for this problem space!
. . . but a lot needs to happen before it becomes feasible.

This experiment has, however clearly demonstrated where the current state's weaknesses are and what needs to be tackled next.

Please feel free to share your Questions, Feedback and Ideas with me now!

Sources

https://kwarc.info/public/proposal_jbetzendahl.pdf
http://dspace.library.uu.nl/handle/1874/396282
https://gl.mathhub.info/MMT/LATIN2/-/tree/devel/elpi
https://github.com/LPCIC/elpi

Full bibliography available upon request.