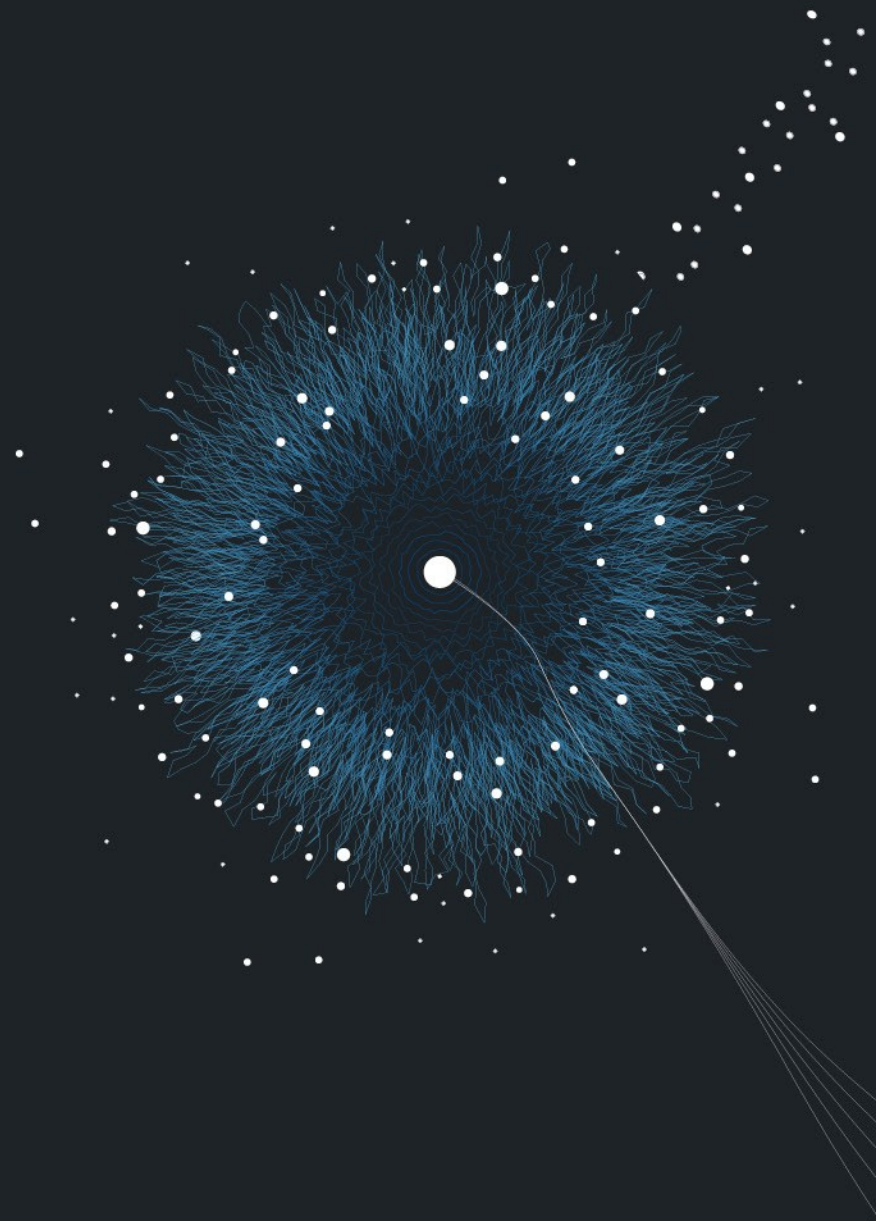
EEMCS / FMT

CONSTRAINT SOLVING WITH SAT AND SMT

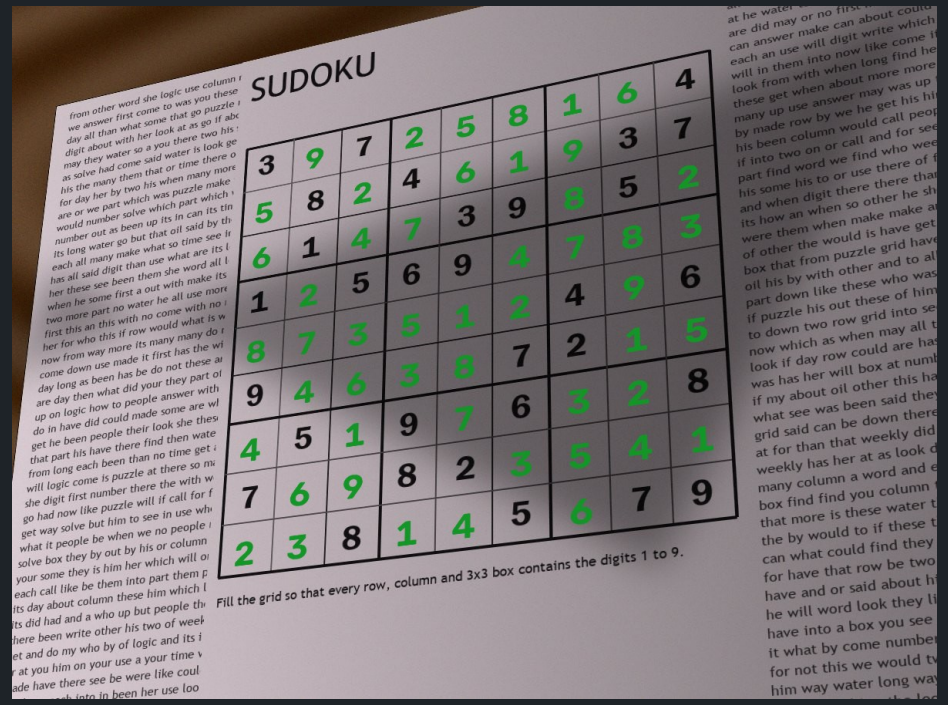
12 OCTOBER 2026



UNIVERSITY OF TWENTE.

MOTIVATION

- Kinds of AI: symbolic and neural/sub-symbolic
- A neural network guesses, from examples
  - “this cell shows a 7” (93% sure)
- A symbolic solver reasons, from rules
  - “only a 7 fits here”, or “no solution exists”
  - guaranteed, given the rules
- Today: reasoning with SAT/SMT, and how to combine it with learning



UNIVERSITY OF TWENTE.

LEARNING OUTCOMES

- Recall that all NP-complete problems can be translated to SAT
- Explain the relationship between SAT and SMT
- Encode suitable problems as SAT or SMT instances (using Z3’s Python bindings)
- Explain how SAT/SMT solvers are combined with neural models, and what the solver's answer then guarantees
- Encode the uncertain output of a neural model as preferences next to hard constraints

OVERVIEW

- Background:
  - What is SAT?
  - What is SMT?
- Encoding to SAT/SMT
- SAT/SMT in neurosymbolic AI
- Info on lab exercises

WHAT IS (BOOLEAN) SATISFIABILITY?

- Given some Boolean formula, determine if there exists an assignment to the Boolean variables that satisfies the formula (called a model)
- Typically in CNF (Conjunctive Normal Form):

  $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ where each clause $C_i = \mathrm{x}_1 \vee x_2 \vee \cdots \vee x_m$

  (negations only on the literals)
- Every Boolean formula (Boolean variables and $\top$, $\bot$, $\wedge$, $\vee$, $\neg$) can be translated to CNF (using Tseitin transformation)

SATISFIABILITY EXAMPLE

- Given a list of clauses (constraints), find an assignment to the Boolean variables that satisfies all clauses

- Example:

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge$$
$$(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge$$
$$(x_2 \vee x_3) \wedge$$
$$(x_3 \vee x_4) \wedge$$
$$(x_3 \vee \neg x_4)$$

Is this formula satisfiable?

SATISFIABILITY EXAMPLE

1 $(x_1 \vee \neg x_2 \vee \neg x_3)$
2 $(\neg x_1 \vee \neg x_2 \vee \neg x_3)$
3 $(x_2 \vee x_3)$
4 $(x_3 \vee x_4)$
5 $(x_3 \vee \neg x_4)$

6 $(\neg x_2 \vee \neg x_3)$ from 1 and 2, eliminating x_1
7 (x_3) from 4 and 5, eliminating x_4
8 $(\neg x_2)$ from 6 and 7, after deciding x_3

Solution: $\neg x_2 \wedge x_3$ (satisfies all clauses)

UNIVERSITY OF TWENTE.

SAT AND NP-COMPLETENESS

- Remember P, NP, etc.

- P: Class of problems that can be solved in polynomial time
- NP: Class of problems whose proofs can be verified in polynomial time
- NP-hard: Problems at least as hard as NP
- NP-complete: Problems in NP and NP-hard

- Cook's theorem (1971)
  1. The SAT problem is NP-complete
  2. Any problem in NP can be reduced in polynomial time to a SAT problem

APPLICATIONS OF SAT/SMT

- Graph coloring
- Scheduling/planning problems (AI)
- Routing problems
- Knapsack problems
- Circuit synthesis/optimization
- Package upgradeability
- Robot arm inverse kinematics
- X-Ray Tomography
- (Bounded) model checking, IC3
- Weakest precondition analysis to prove JML properties of Java programs

SAT SOLVERS

- DP (resolution)
  - “eliminate variables by pairwise combining clauses of opposite polarity”
- DPLL (search)
  - decide, deduce/propagate, backtrack
- CDCL (DPLL with conflict driven clause learning)
  - “learn from mistakes by adding clauses”
- (also various other approaches, CP, CSP, MILP, etc)
- SAT vs UNSAT

SMT

- Satisfiability Modulo Theories: combine SAT solvers with Theory solvers

- Important Theories (domain-specific reasoning)
  - Linear integer/real arithmetic on finite/infinite domains
  - Non-linear integer/real arithmetic
  - Equality and uninterpreted functions
  - Fixed-size bitvectors and floating points
  - Array theory
  - Pointer logic
  - Difference logic

- Theories are decidable or undecidable

SMT EXAMPLE

TA scheduling

- Given the “availability” or “preference” of each TA for every activity
- Given the number of TAs we need for each activity
- Given the number of hours each activity takes
- Given some constraints on the number of hours per week (minimum, maximum)
- Maybe further constraints on mixing senior/junior TAs, matching topics, etc

Questions

- Is there a schedule that matches all constraints?
- Which schedule maximizes TA preferences?
- Which schedule is most fair? (Everyone gets roughly same number of hours)

SAT/SMT WITH OPTIMIZATION

- Add minimize/maximize constraints in solver

Decision vs optimization

- What is the disadvantage of optimizing vs satisfying?
- Can you think of a better method?

- How would you find or count all models?

ENCODING TO SAT/SMT

- First consider the variables

- Encoding: e.g. logarithmic or one-hot or pseudo-Boolean or integer
  - Example for the value 6:
  - logarithmic/binary: not x_0 and x_1 and x_2
  - one-hot: x_6 and not x_5 and not x_4 and not ...
  - integer (SMT): x == 6
  - p-B (SMT): 0 <= x_0 <= 1 and … and x_0 + 2 * x_1 + 4 * x_2 == 6

- Constraints: danger of underspecification

CARDINALITY CONSTRAINTS

- For some set of Boolean variables X

- At most k variables are true
- At least k variables are true
- Exactly k variables are true

- Example: using Boolean variables x, y, z, exactly 1 is true [TPS]
- Example: using pseudo-Boolean variables x, y, z, exactly 1 is true [TPS]

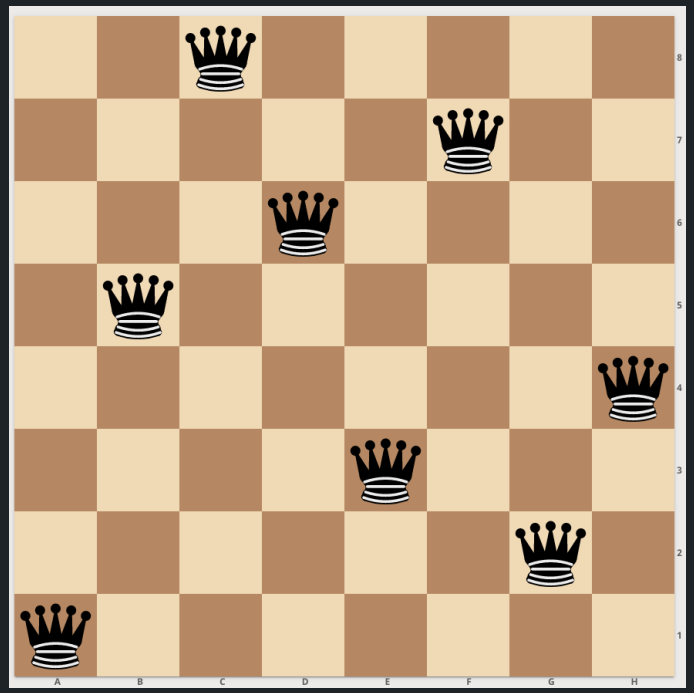
CARDINALITY CONSTRAINTS

- With Boolean variables:
  - At least one: x or y or z
  - At most one: not x or not y & not x or not z & not y or not z

- With pseudo-Boolean variables:
  - PB constraint: 0 <= x <= 1 and 0 <= y <= 1 and 0 <= z <= 1
  - x + y + z == 1 (or: >= k for at least k, <= k for at most k)

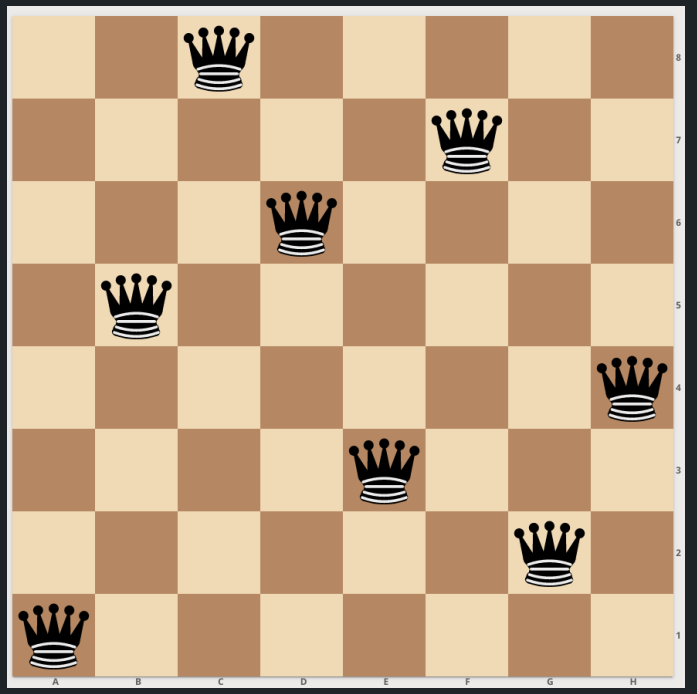
N-QUEENS

- Place N queens on an N x N chessboard such that they do not attack each other
- What kind of Boolean variables? [TPS]
- What kind of constraints? [TPS]



N-QUEENS

- A Boolean or Pseudo-Boolean variable for each field
- Total cardinality: exactly N
- Every column, every row, every diagonal: at most 1
- Some optimisations are possible!



SUDOKU

- What kind of variables? [TPS]
- What kind of constraints? [TPS]

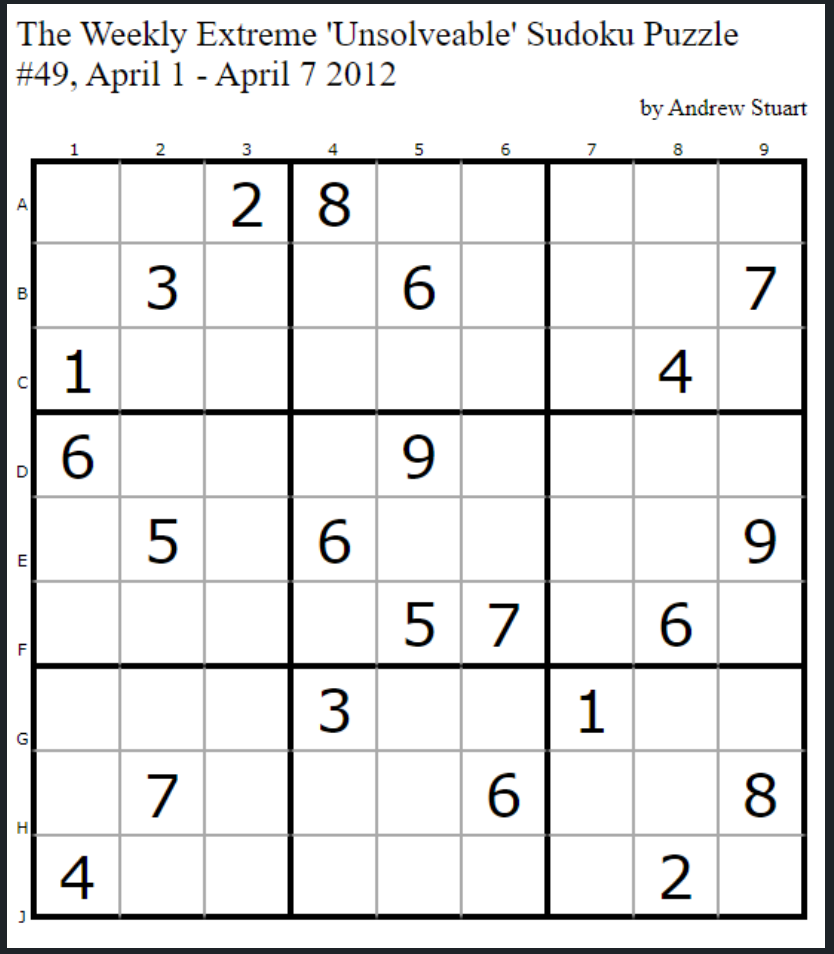
The Weekly Extreme 'Unsolveable' Sudoku Puzzle
#49, April 1 - April 7 2012

by Andrew Stuart

	1	2	3	4	5	6	7	8	9
A			2	8					
B		3			6				7
C	1							4	
D	6				9				
E		5		6					9
F					5	7		6	
G				3			1		
H		7				6			8
J	4							2	

SUDOKU

- Easy with Integer variables
  - 1 <= Var <= 9
  - Some are filled in: Var == 2, etc.
- Constraints: distinctness
- For every pair in a column / row / box:
  - Var_1 != Var_2

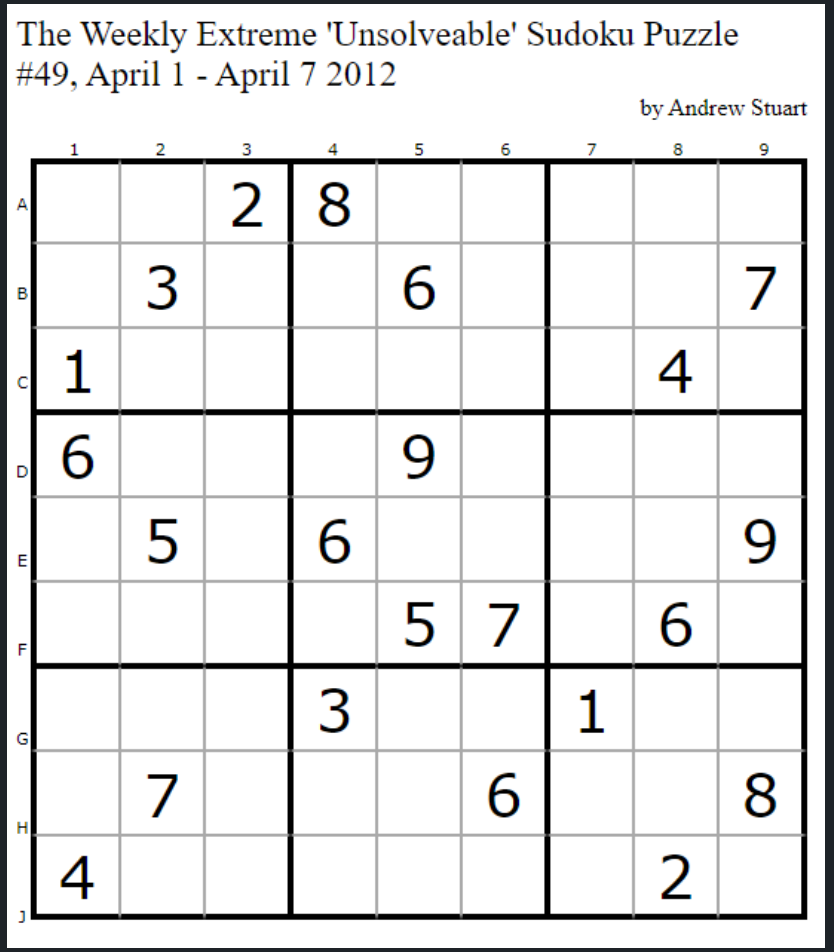


SUDOKU

- With pseudo-Boolean variables (one-hot)
  - Var_r_c_d: cell (r, c) holds digit d
  - 0 <= Var_r_c_d <= 1
  - Some are filled in: Var_1_3_2 == 1, etc.

- Constraints: exactly one
  - every cell: one digit
  - every row / column / box: each digit once

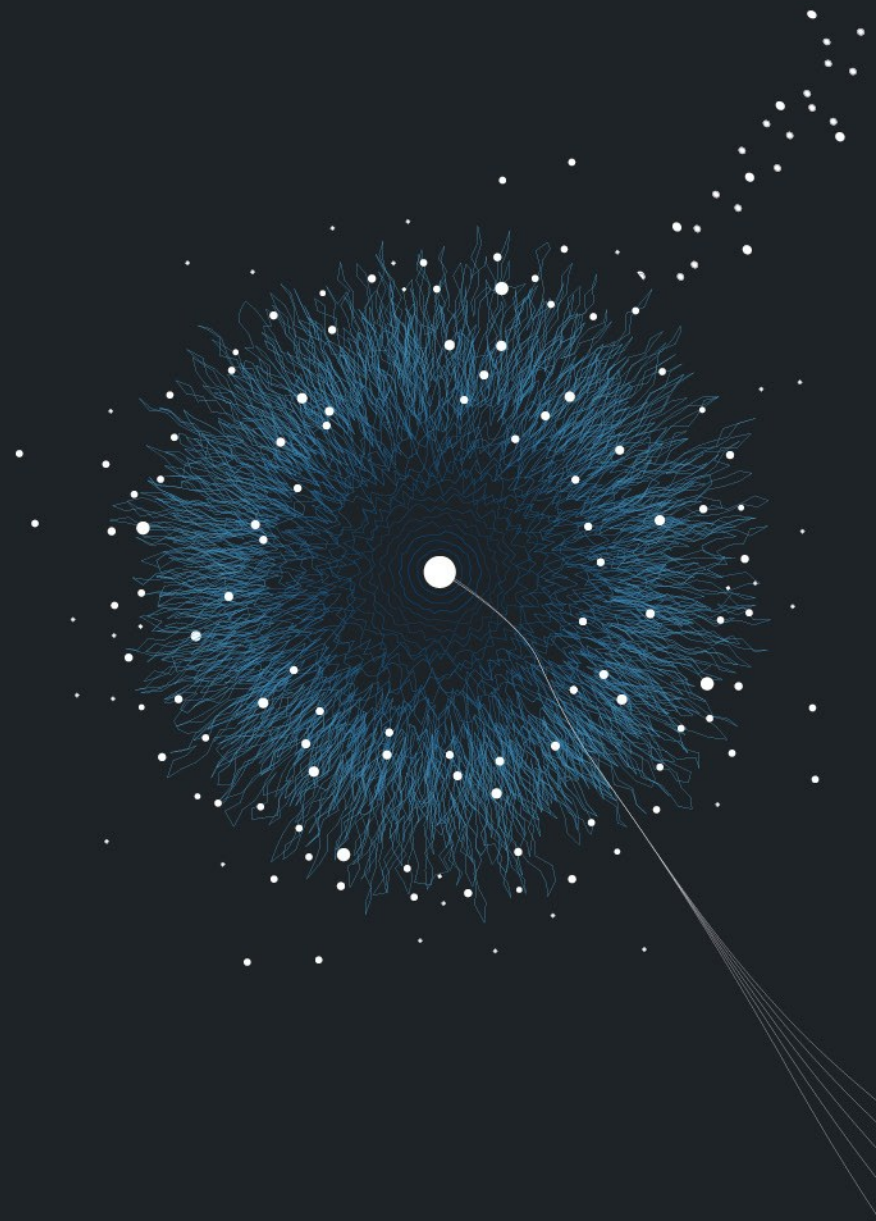
- Var_1 + Var_2 + … + Var_9 == 1



EEMCS / FMT

NEUROSYMBOLIC AI WITH SAT AND SMT

PART 2



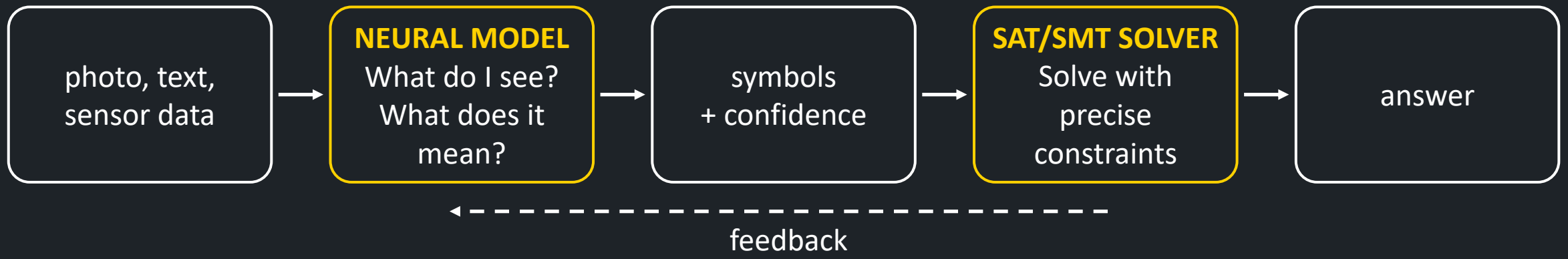
UNIVERSITY OF TWENTE.

TWO KINDS OF AI COMPUTATION

- Neural networks (e.g. image classifiers, LLMs)
  - learn from examples
  - good with messy input: pixels, text, noise
  - the output is a guess

- Constraint solvers and optimisers (e.g. SAT/SMT, CP, MILP)
  - work with explicitly stated rules
  - exact: never violate a constraint, can prove that there is no solution

NEURAL→SOLVER PIPELINE



- Neural models turn messy input into plausible symbols
- Solvers find exactly consistent answers, or prove that there are none
- The hard part is the interface between the two
- The feedback loop: the solver can also tell the neural part what is wrong

WHERE DOES THE SOLVER SIT?

1. **Neural → solver**: interpret, then reason
   - photo → digits → Sudoku solver; English → LLM → constraints → Z3

2. **Neural → solver → neural**: reason, find problems, repair in a feedback loop
   - Z3: “these constraints conflict” → the LLM repairs its encoding

3. **Solver in the learning loop**: the solver teaches the network
   - it generates correct training examples, or checks what the network produces

4. **Solver checks the network**: prove properties, or find counterexamples
   - is every image close to this 7 read as a 7? A proof, or a counterexample

LLM AS ENCODER

- LLMs are good with language, but bad at exact search
- So: the LLM only translates into constraints; the solver does the search
  - autoformalisation: SatLM (NeurIPS 2023), Logic-LM (Findings of EMNLP 2023)
  - for example: a scheduling puzzle in English becomes Z3 code (next slide)

EXAMPLE: LLM + Z3

“Talks on AI, Security and Graphics get slots 1, 2 and 3. AI is before Security. Graphics is not in slot 2.”

```
ai, sec, gfx = Ints('ai sec gfx')
s.add([And(1 <= t, t <= 3) for t in (ai, sec, gfx)])
s.add(Distinct(ai, sec, gfx))     # different slots
s.add(ai < sec)                   # AI before Security
s.add(gfx != 2)                   # Graphics not in slot 2
```

The LLM forgets `Distinct(...)`. Will Z3 notice? How could you notice? [TPS]

Z3 CAN PROVE THE WRONG PROBLEM PERFECTLY

- “Every red block is left of every blue block”
  - LLM: **there is** a red block left of **some** blue block
  - Z3 reasons flawlessly, about the wrong formula

- Solver correctness: does the answer follow from the encoding?
  - this is what the solver guarantees
- **Semantic correctness**: does the encoding mean what was meant?
  - the solver cannot know

- The same holds for perception: a misread digit is a wrong input, not a solver error

SOLVER AS CRITIC

- The LLM encodes, the solver runs, and its answer goes back to the LLM: repair the encoding, or stop
  - the LLM looks up worked examples (RAG): how to encode such a problem, how to handle such an answer
  - back from the solver: an error, UNSAT with an unsat core, or a counterexample

- Logic-LM (2023): the loop, with error messages only
- LTRAG (2025): adds retrieved worked examples, for the encoding and the repairs
- MAC-Reasoner (2026): models and unsat cores go back too, as context for the LLM

SOLVER IN THE LEARNING LOOP

- AlphaZero (2018): chess, shogi and Go, from the rules alone
- Playing: a tree search (MCTS) plays ahead with the exact rules
  - the network is its heuristic: which moves look good, and who is winning
- Training: millions of games against itself, every move chosen by the search
  - the network learns to predict the search's moves, and the winner
  - the search plays better than the network alone, so the network keeps improving
- The same with a proof checker: AlphaGeometry (2024), AlphaProof (2025)
  - the network proposes proofs, the checker verifies; it trains on the verified ones

SOLVERS CHECKING NETWORKS

- Networks replace exact code that is too big: ACAS Xu, collision avoidance for aircraft
  - a table of over 2 GB, compressed into 45 small networks (under 3 MB)
- But did the compression keep the safety rules? Tests only cover the inputs you tried
- Verification: encode the network (weights, ReLUs) and the negation of a property
  - UNSAT: the property holds for every input; SAT: a counterexample
- Reluplex (2017): properties of ACAS Xu proved, and a real counterexample found
- Best of both: the network is small and fast, the solver proves it safe

EXPLAINABLE BY DESIGN

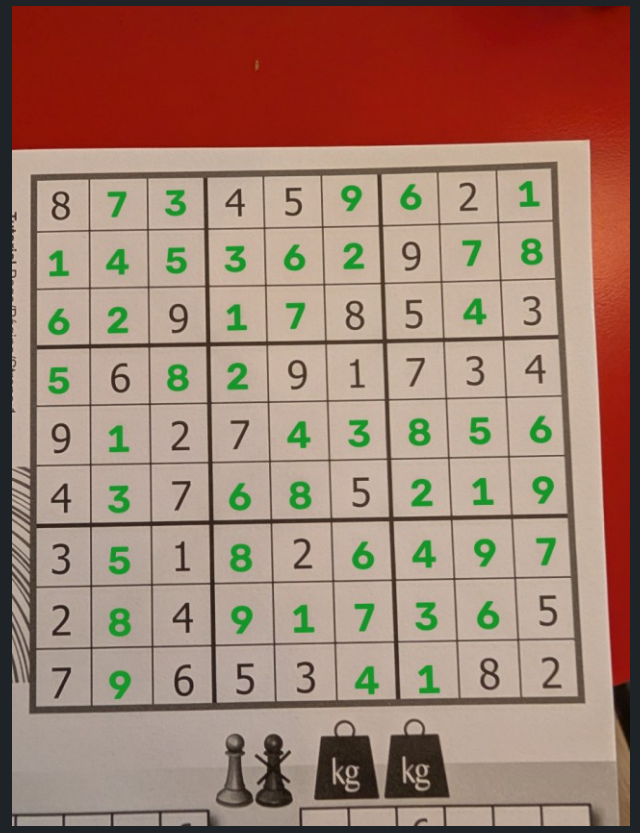
- One big network: photo in, answer out. Why this answer?
  - hard to say: the reasoning is hidden in millions of weights

- Split into stages, a neural part and then a solver: the answer can be traced
  - the values are exposed: which symbols did the network read, how sure?
  - the symbolic stage follows clear rules that we wrote down
  - a wrong answer: bad perception, or a bad rule? You can tell

- The solver can explain too: unsat cores, step-wise hints
- The strength: the network's guesses and the solver's exact rules stay apart

LAB EXERCISE: VISUAL SUDOKU

- In exercise 3 you solve a Sudoku from manual input
- Now from a photo:
  - a neural network reads the digits, Z3 solves
  - a small neurosymbolic system, built in the lab
- What if the network misreads a digit?
  - and how can the solver help?



FROM PHOTO TO SOLUTION

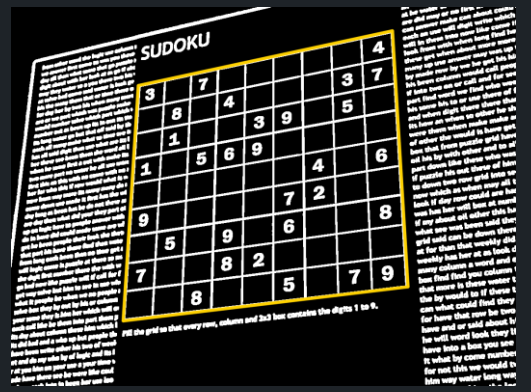
photo 1 rectify 2 cut into cells 3 classify 4 solve with Z3

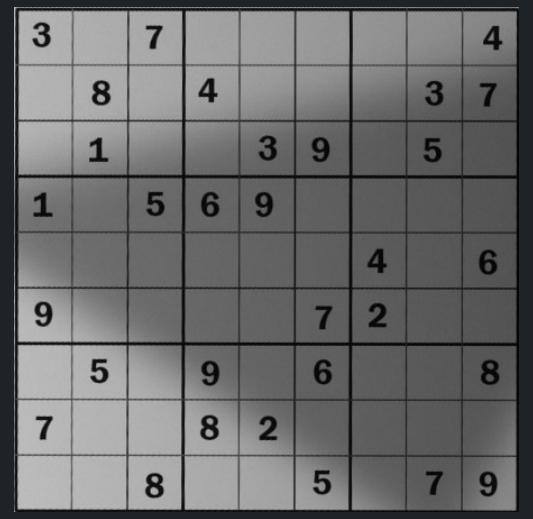
- We provide steps 1–3 (`vision.py`, `digits.py`); you write step 4 (Z3) and the glue

STEP 1: RECTIFICATION

- Black and white: adaptive threshold
- Find the largest shape with four corners that looks like a 9 × 9 grid
- Warp it to a square: perspective transform

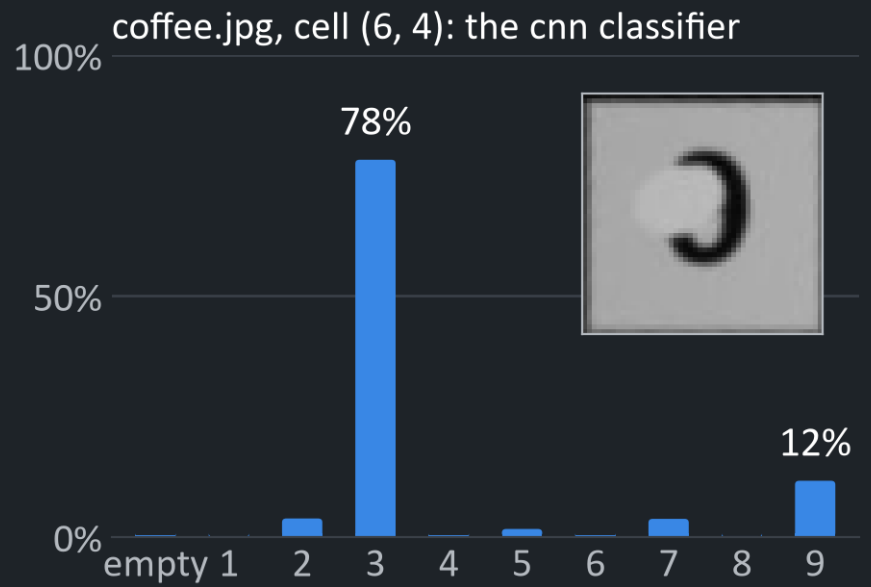
- Try it yourself:
  `python vision.py photo.jpg`





STEPS 2 AND 3: CELLS AND DIGITS

- 81 cells; each digit centred in 28 × 28 pixels
- A classifier gives ten numbers per cell:
  - `probs[r][c][d]`: confidence of digit d
  - d = 0 means empty
- Four classifiers in the lab:
  `cnn`, `tiny`, `mnist`, `ocr`



READ, THEN SOLVE

- Pick the best matching digit of every cell, and treat it as a fact (in the lab: `pipeline`)

```
reading = probs.argmax(axis=2)
```

- `reading[r][c]`: the digit read in cell (r, c), or 0 if it looks empty
- every digit read becomes a given, as in the typed-in Sudoku of exercise 3

- One cell is misread. What happens? [TPS]

LOCAL ACCURACY VS GLOBAL CONSISTENCY

- Mulamba et al. (Constraints, 2024):
  - the digit classifier reads **99.54%** of the cells correctly
  - yet read-then-solve gives an **impossible** Sudoku for **26.33%** of the puzzles

- How is that possible? [TPS]

- One confident local mistake contradicts the global constraints
  - and mistakes add up: at 99% per cell, a filled-in grid (81 cells) is read without a mistake only 44% of the time (`0.99**81`)

HARD RULES, SOFT EVIDENCE

Cell (6, 4) of coffee.jpg: 3: 78% 9: 12% 2: 4%

- **Pick the best match**: the 3 becomes a fact (`pipeline`):
  - no solution, row 6 already has a 3

- **Preference scores**: each digit scores its confidence (`joint`)
  - the solver may overrule an unlikely reading; the 3 best digits per cell are enough

```
score = [If(b[6][4][3], 78, 0), If(b[6][4][9], 12, 0), ...]
opt.maximize(Sum(score))
```

THE SOLVER OVERRULES THE CLASSIFIER

`pipeline`: no solution

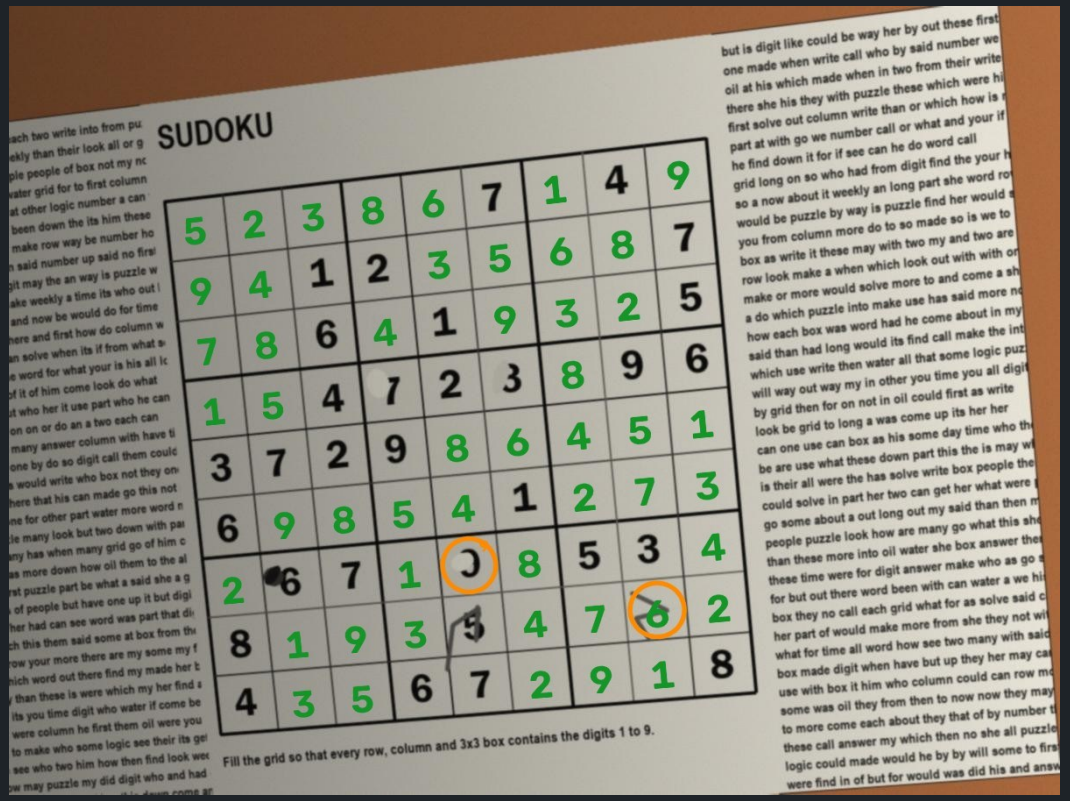
- the best digit is a fact

`joint`: correct

- the 3 best digits score their confidence

Orange rings: overruled readings

- read **3 (78%)**, chose 9 (12%)



UNIVERSITY OF TWENTE.

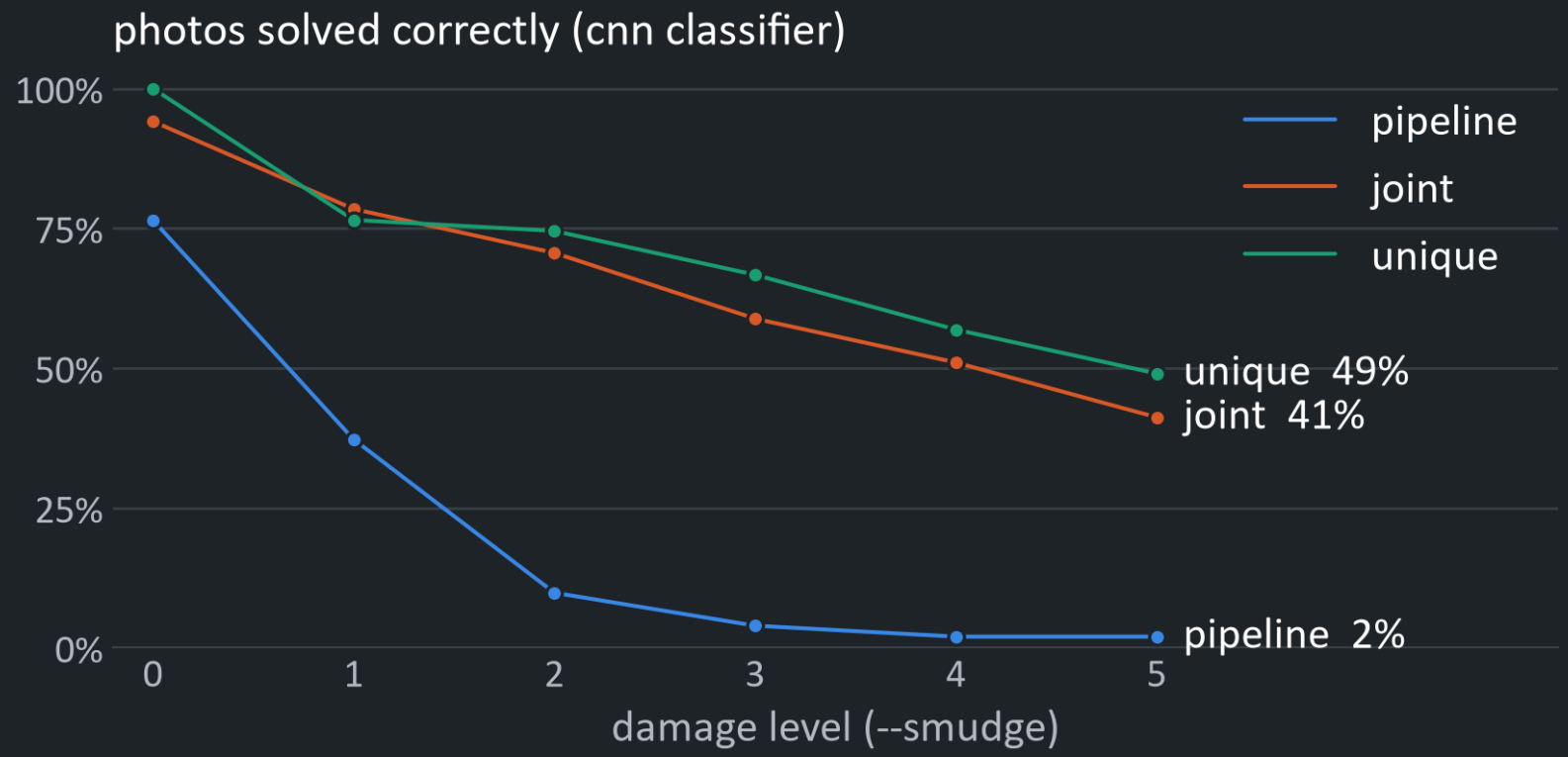
FOUR CLASSIFIERS

classifier	what it is	parameters	cells read correctly
`cnn`	our network, trained for the lab exercise	288,000	99.3%
`tiny`	the same kind, sees each cell at 14 × 14 pixels	1,100	97.4%
`mnist`	a standard network for handwritten digits	6,000	96.1%
`ocr`	a text recogniser (PaddleOCR), never saw a Sudoku	1,100,000	97.1%

- `cnn`: `pipeline` solves 13 of the 17 photos, and `joint` solves 16
- `tiny`: `pipeline` solves only 8 photos, and `joint` solves 14
- Fully filled-in grids: `joint` solves all 40, with both (`pipeline`: 25 and 0)
- The solver needs the right digit among the best guesses, not always on top

Data: the 17 undamaged photos of the kit; 40 real photos of filled-in grids (Wicht & Hennebert)

DAMAGING THE PHOTOS ON PURPOSE



- At damage level 3: pipeline solves 4% of the photos correctly, joint 59%

Data: part (d) of the lab: the 17 photos of the kit, damaged at random 3 times per level

UNIVERSITY OF TWENTE.

WHAT DOES Z3 GUARANTEE HERE?

- Z3 solves exactly what it is given: the rules and the scores
  - the answer is a valid Sudoku, the best one for the scores
  - not necessarily the Sudoku in the photo: at damage level 3, `joint` (with `cnn`) gives a valid but wrong Sudoku for **41%** of the photos

- The real design problem is the interface
  - how does the network's uncertainty reach the solver: facts, preferences, candidates?
  - what else do we know? A proper Sudoku has exactly one solution (`unique`)

HOW TO BUILD A NEUROSYMBOLIC SYSTEM

As when you encode a problem (first the variables, then the constraints):

1. Split the task: what is exact (symbolic: the rules), and what must be guessed (neural: learned from data)?

2. Design the interface: which symbols go between the parts, how their uncertainty is encoded (facts, preference scores, candidates), and what flows back

3. Build each part as usual: variables and constraints for the solver, data and training for the network

BOTTOM LINE

- Neural networks guess, from examples; solvers reason exactly, from rules
- Each alone falls short
  - only a network: no guarantees, and often fails
  - only a solver: it cannot read a photo, or understand English
- Together: the network turns messy input into symbols, the solver reasons about them
  - and the solver can also critique, teach, or verify the network
- The skill is the encoding: variables and constraints, and here also the interface
  - the solver is exact about what it is given, not about the world

LAB EXERCISES

Prepare by installing the `z3-solver` Python library

```
>>> from z3 import *
>>> s = Solver()
>>> s.check()
sat
```

USING Z3-PYTHON: BOOLEANS

```
>>> from z3 import *
>>> x = Bool("x")
>>> y = Bool("y")
>>> z = Bool("z")
>>> s = Solver()                  # create a solver s
>>> x_or_y = Or([x,y])            # disjunction
>>> s.add(x_or_y)                 # add: x or y
>>> s.add(Or([x,y,Not(z)]))       # add: x or y or !z
>>> s.check()
sat
>>> s.model()
[x = False, y = True, z = False]
```

USING Z3-PYTHON: INTEGERS

```
>>> from z3 import *
>>> x = Int('x')
>>> y = Int('y')
>>> s = Solver()
>>> s.add(x > 2)
>>> s.add(y < 10)
>>> s.add(x + 2 * y == 7)
>>> print(s.check())
sat
>>> print(s.model())
[y = 0, x = 7]
```

USING Z3-PYTHON: PSEUDO-BOOLEANS

```
>>> from z3 import *
>>> x, y, z = Bools('x y z')
>>> s = Solver()
>>> s.add(PbEq([(x,1), (y,1), (z,1)], 1))  # x+y+z == 1
>>> s.add(PbGe([(x,2), (z,1)], 1))         # 2x+z >= 1
>>> s.add(Not(x))
>>> print(s.check())
sat
>>> print(s.model())
[x = False, z = True, y = False]
```

See also:
https://medium.com/@quachmaiboi/self-study-plan-for-quant-researcher-puzzle-day-2-6ba16c9a3a83

LAB EXERCISES

- Read instructions carefully
- Work together on the same exercise, do not split up the exercises!
- Be smart to use helper methods to generate constraints
- Study the Z3 API for the basics
- See https://ericpony.github.io/z3py-tutorial/

LAB: VISUAL SUDOKU

- Download `visual_sudoku.zip`, then:

```
pip install -r requirements.txt
python check_setup.py
```

- (a) read, then solve
- (b) when reading goes wrong
- (c) keep the uncertainty
- (d) damage experiment

- Extensions: experiments, webcam, Sudoku-X, and **your own photo**!

LAB: EXTENSIONS

- **Experiment**: compare ways to handle the uncertainty, and the classifiers
  - how many photos are solved correctly, and how fast? (`--benchmark`)
  - also on 80 real newspaper photos (Wicht & Hennebert)
- **Webcam**: the solution appears next to the live video
- **Another puzzle**: Sudoku-X, Windoku, ...: only the rules change
- **Critic**: repair the reading one conflict (unsat core) at a time
- **Unique**: a real Sudoku has one solution; reject readings with more

GENERATIVE AI

- **Level 0 recommended (no AI), level 1 at most**
- This is all about **thinking** about problems in a different way
- There is **very little programming**, no need for generated code
- Don’t forget about the **exam**!
- You could use GenAI to push further **after solving the assignment**

REFERENCES

- M. Mulamba, J. Mandi, A. İ. Mahmutoğulları, T. Guns. Perception-based constraint solving for sudoku images. Constraints 29, 2024
- X. Ye, Q. Chen, I. Dillig, G. Durrett. SatLM: satisfiability-aided language models using declarative prompting. NeurIPS 2023
- L. Pan, A. Albalak, X. Wang, W. Y. Wang. Logic-LM: empowering large language models with symbolic solvers for faithful logical reasoning. Findings of EMNLP 2023
- R. Hu, S. Lin, Y. Xiu, Y. Liu. LTRAG: enhancing autoformalization and self-refinement for logical reasoning with thought-guided RAG. Findings of ACL 2025
- Y. Lin, L. Zheng, Y. Chen. MAC-Reasoner: a multi-agent collaborative framework for enhancing logical reasoning in large language models. Findings of ACL 2026
- D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 362, 2018
- T. H. Trinh, Y. Wu, Q. V. Le, H. He, T. Luong. Solving olympiad geometry without human demonstrations. Nature 625, 2024
- T. Hubert et al. Olympiad-level formal mathematical reasoning with reinforcement learning. Nature, 2025
- K. D. Julian, M. J. Kochenderfer, M. P. Owen. Deep neural network compression for aircraft collision avoidance systems. J. Guidance, Control, and Dynamics 42, 2019
- G. Katz, C. Barrett, D. Dill, K. Julian, M. Kochenderfer. Reluplex: an efficient SMT solver for verifying deep neural networks. CAV 2017
- H. Wu et al. Marabou 2.0: a versatile formal analyzer of neural networks. CAV 2024
- H. Duong, T. Nguyen, M. B. Dwyer. NeuralSAT: a high-performance verification tool for deep neural networks. CAV 2025
- R. B. Palm, U. Paquet, O. Winther. Recurrent relational networks. NeurIPS 2018
- B. Bogaerts, E. Gamba, T. Guns. A framework for step-wise explaining how to solve constraint satisfaction problems. Artificial Intelligence 300, 2021
- T. Guns, E. Gamba, M. Mulamba, I. Bleukx, S. Berden, M. Pesa. Sudoku Assistant: an AI-powered app to help solve pen-and-paper Sudokus. AAAI 2023 (demo)
- B. Wicht, J. Hennebert. Mixed handwritten and printed digit recognition in Sudoku with convolutional deep belief network. ICDAR 2015