# Scheduling a small conference with z3 solver

Ben Clifford
benc@hawaga.org.uk
PyBerlin April 2026

# What is z3?

- z3 is SMT solver
- **S**atisfiability
- **M**odulo
- **T**heorems

# ParslFest

- niche academic conference/day
- ~8 years
- ~35 talks, 6 sessions
- 30-100 participants
- manual talk schedule...
  ... a bit too hard to manage all the constraints.
- overengineer a solution!

- pip install z3-solver

# (boolean) satisfiability

```
from z3 import *

x = Bool('x')

y = Bool('y')

solve( And(x, y) )


[y = True, x = True]
```

# (boolean) satisfiability

```
from z3 import *

x = Bool('x')

y = Bool('y')

solve( Or(x, y) )


[y = False, x = True]
```

# Scheduling a tiny conference

- Speaker A, Speaker B
- Session 1, Session 2

```
from z3 import *
speaker_A_in_session_1 = Bool('speaker_A_in_session_1')
speaker_B_in_session_1 = Bool('speaker_B_in_session_1')
speaker_A_in_session_2 = Bool('speaker_A_in_session_2')
speaker_B_in_session_2 = Bool('speaker_B_in_session_2')
```

# Tiny theory of conference scheduling

- All speakers must speak
- Two speakers cannot speak in the same session

# All speakers must speak

```
speaker_A_must_speak = Or(speaker_A_in_session_1, speaker_A_in_session_2)
speaker_B_must_speak = Or(speaker_B_in_session_1, speaker_B_in_session_2)
all_speakers_must_speak = And(speaker_A_must_speak, speaker_B_must_speak)
solve(all_speakers_must_speak)
```

```
[speaker_B_in_session_1 = True,
speaker_A_in_session_2 = False,
speaker_A_in_session_1 = True,
speaker_B_in_session_2 = False]
```

# One speaker per session

```
session_1_one_speaker = Not(And(speaker_A_in_session_1, speaker_B_in_session_1))
session_2_one_speaker = Not(And(speaker_A_in_session_2, speaker_B_in_session_2))
sessions_have_one_speaker = And(session_1_one_speaker, session_2_one_speaker)
conference_schedule_is_valid = And(all_speakers_must_speak, sessions_have_one_speaker)
solve(conference_schedule_is_valid)
```

```
[speaker_A_in_session_2 = True,
 speaker_A_in_session_1 = False,
 speaker_B_in_session_1 = True,
 speaker_B_in_session_2 = False]
```

# Why embed in Python?

- EDSL: embedded domain specific language
- Don’t reinvent useful things
- Python good for (e.g.)
  - generating constaints (e.g. read yaml file)
  - formatting output (e.g. present as a schedule)

# Scheduling ParslFest

- 35 speakers in 6 sessions over 2 days
- Sessions contain multiple speakers
- Several ad-hoc constraints, e.g.
  - remote speaker timezones
  - childcare
  - related talks

# Theory of Bit Vectors

- before: `x = Bool('x')` # True or False
- `x = BitVec('x', 4)` # a 4-bit number, 0..15
- Theory of 4-bit integers/bit vectors
- ParslFest schedule solution will be:
  - one variable per speaker (35 variables)
  - BitVec(4) listing their session number (1-6)

# Speakers in valid sessions

```
conference_schedule_is_valid = []
speakers = {}
for n in range(1,36):
  speakers[n] = BitVec(f'speaker_{n}_in_session', 4)
  conference_schedule_is_valid.append(
    And(speakers[n] >= 1, speakers[n] <= 6)
  )
solve(And(*conference_schedule_is_valid))
```

# Speakers in valid sessions

```
[speaker_20_in_session = 4,
 speaker_34_in_session = 4,
 speaker_9_in_session = 4,
 speaker_35_in_session = 4,
 speaker_15_in_session = 4,
 speaker_13_in_session = 4,
 speaker_28_in_session = 4,
 speaker_14_in_session = 4,
 speaker_29_in_session = 4,
 speaker_17_in_session = 1,
 speaker_10_in_session = 4,
 speaker_7_in_session = 4,
 speaker_12_in_session = 4,
 speaker_32_in_session = 4,
 speaker_11_in_session = 4,
 speaker_19_in_session = 4,
 speaker_16_in_session = 4,
 speaker_16_in_session = 4,
 speaker_21_in_session = 4,
 speaker_26_in_session = 4,
 speaker_31_in_session = 4,
 speaker_22_in_session = 4,
 speaker_27_in_session = 4,
 speaker_5_in_session = 4,
 speaker_33_in_session = 4,
 speaker_4_in_session = 4,
 speaker_8_in_session = 4,
 speaker_25_in_session = 4,
 speaker_2_in_session = 4,
 speaker_18_in_session = 4,
 speaker_1_in_session = 4,
 speaker_23_in_session = 4,
 speaker_3_in_session = 4,
 speaker_24_in_session = 4,
 speaker_6_in_session = 4,
 speaker_30_in_session = 4]
```

# Sessions are filled evenly

35 talks in 6 sessions

=> 5.8333... talks per session

=> constraint: 5 or 6 talks per session

# And/Or/...

- And(statements) # all of these statements must be true
- Or(statements) # at least one of these statements must be true

- AtMost(statements, n) # at most n statements can be true
- AtLeast(statements, n) # at least n statements must be true

# Sessions are filled evenly

```
for s in range(1,7):
  conference_schedule_is_valid.append(
    AtMost(*[speakers[n] == s
             for n in range(1,36)],
           6))
```

6 AtMost constraints, each containing 35 rules

AtLeast constraints look similar

# Sessions are filled evenly

```
[speaker_4_in_session = 6,
 speaker_31_in_session = 6,
 speaker_17_in_session = 6,
 speaker_11_in_session = 6,
 speaker_15_in_session = 5,
 speaker_30_in_session = 2,
 speaker_1_in_session = 3,
 speaker_8_in_session = 5,
 speaker_22_in_session = 2,
 speaker_19_in_session = 5,
 speaker_10_in_session = 6,
 speaker_13_in_session = 4,
 speaker_20_in_session = 4,
 speaker_23_in_session = 2,
 speaker_25_in_session = 1,
 speaker_27_in_session = 2,
 speaker_16_in_session = 4,
 speaker_7_in_session = 3,
 speaker_28_in_session = 1,
 speaker_34_in_session = 4,
 speaker_3_in_session = 5,
 speaker_2_in_session = 4,
 speaker_5_in_session = 3,
 speaker_21_in_session = 2,
 speaker_12_in_session = 1,
 speaker_14_in_session = 5,
 speaker_9_in_session = 3,
 speaker_24_in_session = 6,
 speaker_18_in_session = 1,
 speaker_33_in_session = 1,
 speaker_26_in_session = 3,
 speaker_32_in_session = 4,
 speaker_29_in_session = 3,
 speaker_35_in_session = 2,
 speaker_6_in_session = 1]
```

what I’ve done so far is ...

... a complicated way of dividing a list of 35 items into 6 chunks

... but now easy to add more interesting constraints

# More interesting constraints

- Speakers only present on day 1 or day 2 (sessions 1-3 or 4-6)
- Two related talks should be scheduled in same session
- One human speaks twice - prefer on different days
- Balance of remote and in-person talks per sessions
- Group related topics into same session

These were hard to track manually

# example individual constraint 1

- Speaker 4:
  “I can only speak in the morning of day 1”
- Speaker 4 can only speak in sessions 1 and 2
- `conference_schedule_is_valid.append(`

```
    Or(speakers[4] == 1, speakers[4] == 2))
```

# example individual constraint 2

- Speaker 23 and speaker 24 are giving two-part talk
- Speaker 23 and speaker 24 should be in the same session
- `conference_schedule_is_valid.append(`

  `speakers[23] == speakers[24])`

# Soft constraints

- constaints we would like to be true, but it is ok if they can’t be.

# Soft constraints

```
opt = Optimize()
opt.add(And(*conference_schedule_is_valid))

# these can't both be true
opt.add_soft(speakers[1] != 5)    # speaker 1 is not in session 5
opt.add_soft(speakers[1] == 5)   # speaker 1 is in session 5

opt.check()
m=opt.model()
print(m)
```

# loose grouping of topics

- “nice to have”  - group talks by topic
- Too hard for lazy humans
- Other constraints more important

# example: group prime-numbered speakers

```
primes = [2,3,5,7,11,13,17,19,23,29,31]

for a in primes:

  for b in primes:

    if a > b:

      opt.add_soft(speakers[a] == speakers[b], id="primegroup")
```

not possible to fully solve: too many to fit in one 6 talk session

Increased runtime: 5 minutes vs <1s

... but still usable for scheduling ParslFest

# final schedule

*** Session 1
Speaker 1 <- violated soft constraint (speaker 1 in session 5)
Speaker 4  <- speaker 4 must be in session 1 or 2
Speaker 6
Speaker 9
Speaker 15
Speaker 33
*** Session 2
Speaker 3 [PRIME]
Speaker 5 [PRIME]
Speaker 7 [PRIME]
Speaker 17 [PRIME]
Speaker 29 [PRIME]
Speaker 31 [PRIME]
*** Session 3
Speaker 12
Speaker 14
Speaker 16
Speaker 25
Speaker 32
*** Session 4
Speaker 10
Speaker 18
Speaker 21
Speaker 28
Speaker 30
Speaker 34
*** Session 5
Speaker 8
Speaker 20
Speaker 22
Speaker 26
Speaker 27
Speaker 35
*** Session 6
Speaker 2 [PRIME]
Speaker 11 [PRIME]
Speaker 13 [PRIME]
Speaker 19 [PRIME]
Speaker 23 [PRIME]
Speaker 24  <- 24 has to be with speaker 23

real  5m19.103s

# Pretty print the schedule

```
opt.check()

m=opt.model()

for session in range(1,7):
  print(f"*** Session {session}")
  for n in range(1,36):
    if m.evaluate(speakers[n]) == session:
      print(f"Speaker {n}")
```

# Real ParslFest 2025 schedule

- balanced session size
- speaker personal constraints
- session chairs
- stickiness once speaker told session

Soft constraints:

- Balance remote vs in-person per session
- 19 topics

These soft constraints are very tricky: 66 minutes to schedule

(if I worked on this more, I'd work on the theory around these constraints)

# Other examples

- software verification / theorem proving
- https://www.keiruaprod.fr/blog/2021/05/09/z3-samples.html
  - puzzles: xkcd 287, sudoku
  - resource allocation
  - tax law / tax optimisation
- Factory layout
  http://theory.stanford.edu/~nikolaj/supercharging.html
  1000 operators, 10000 tasks, 1000 tools

# summary

- satisfiability - boolean statements
- theories:
  - built in - bit vectors
  - added on - toy conference rules
  - z3 implements others, eg: floats, strings, arrays, regexp
  - user pluggable “propagators” for new theories
- embedded in Python - useful pre/post processing
- scales to large problems - even though this was pretty small
- human interaction:
  - more declarative than writing procedural Python
  - represent logical rules, don’t represent search algorithm
- mandatory AI mention: reminiscent of the 1970s/1980s Prolog 5$^{th}$ generation AI movement