

Clauses and Beyond: On Fast Prototyping with SAT Oracles

Alexey Ignatiev

September 8, 2021



Motivation



The SAT disruption

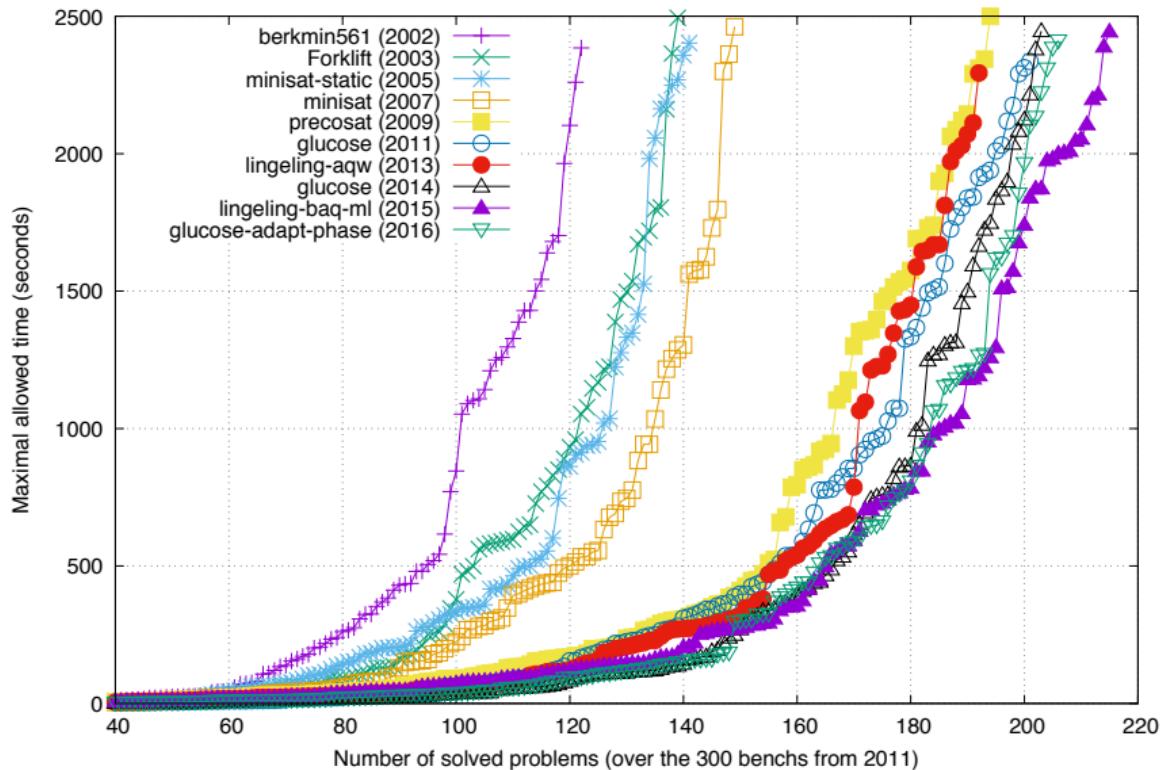
- **SAT is NP-complete**

- SAT is NP-complete
 - But CDCL SAT solving is a **success story** of Computer Science
 - **Hundreds (thousands?) of practical applications**



SAT solver improvement

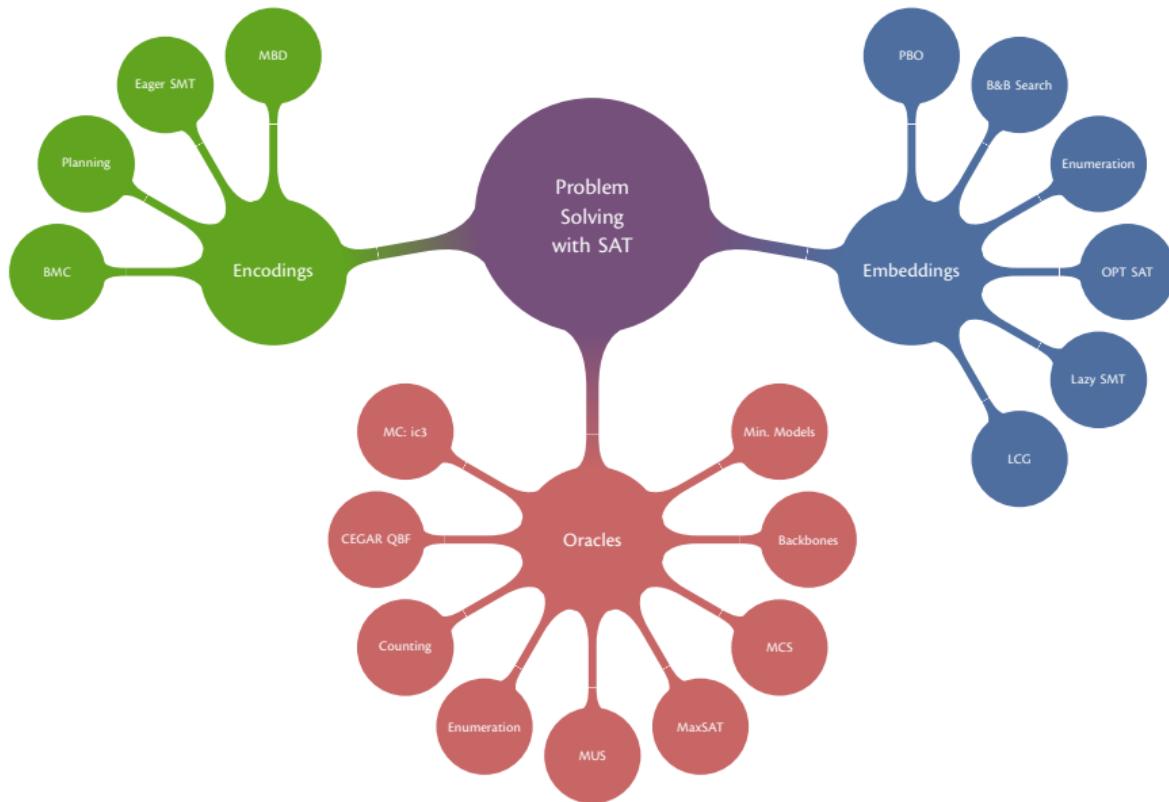
[Source: Simon 2015]



SAT is the engines' engine



SAT is ubiquitous in problem solving

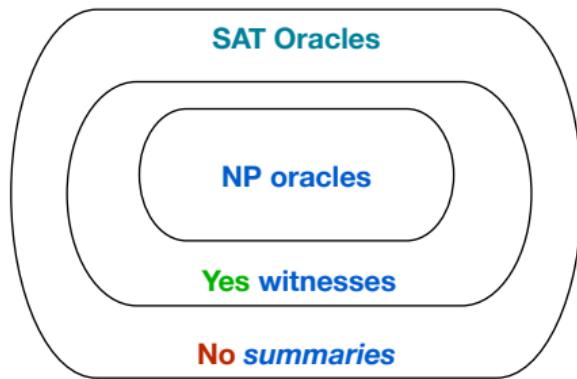


How to solve problems with SAT?

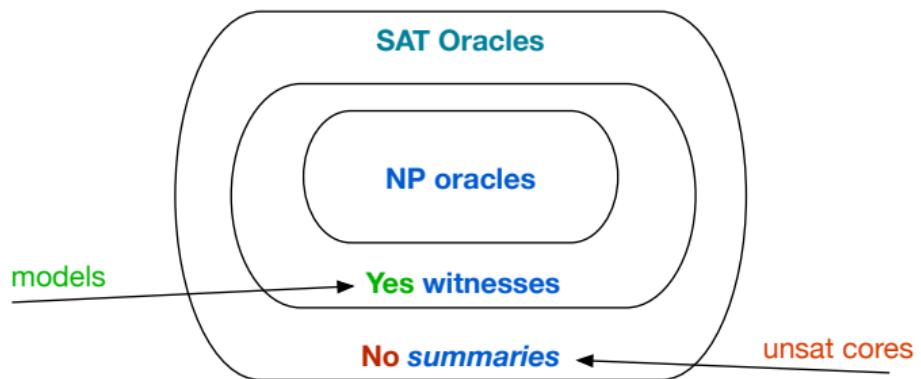
- Use SAT solvers as oracles
- Should be quick to prototype
- Should be reasonably efficient
- Should enable fiddling with the algorithms
- Avoid steep learning curves
- Combine with other technologies
- ...

SAT Oracles

What are SAT oracles?



What are SAT oracles?



Where are we using SAT oracles?

MaxSAT

Where are we using SAT oracles?

MaxSAT

MinSAT; Maximal
Falsifiability

Where are we using SAT oracles?

MaxSAT

MCS Extraction
& Enumeration

MinSAT; Maximal
Falsifiability

Where are we using SAT oracles?

MaxSAT

MCS Extraction
& Enumeration

MUS Extraction
& Enumeration

MinSAT; Maximal
Falsifiability

Where are we using SAT oracles?

MaxSAT

MCS Extraction
& Enumeration

MUS Extraction
& Enumeration

MinSAT; Maximal
Falsifiability

MSMP

Where are we using SAT oracles?

MaxSAT

MCS Extraction
& Enumeration

MUS Extraction
& Enumeration

MinSAT; Maximal
Falsifiability

Lean Kernels;
Backbones

MSMP

Where are we using SAT oracles?

MaxSAT

MCS Extraction
& Enumeration

MUS Extraction
& Enumeration

MinSAT; Maximal
Falsifiability

Lean Kernels;
Backbones

MSMP

Function Minimization; Prime Enumeration; Propositional Abduction;

Where are we using SAT oracles?

MaxSAT	MCS Extraction & Enumeration	MUS Extraction & Enumeration
MinSAT; Maximal Falsifiability	Lean Kernels; Backbones	MSMP
Function Minimization; Prime Enumeration; Propositional Abduction;		
Model-based Diagnosis; Axiom Pinpointing; Package Management;		

Where are we using SAT oracles?

MaxSAT

MCS Extraction
& Enumeration

MUS Extraction
& Enumeration

MinSAT; Maximal
Falsifiability

Lean Kernels;
Backbones

MSMP

Function Minimization; Prime Enumeration; Propositional Abduction;

Model-based Diagnosis; Axiom Pinpointing; Package Management;

QBF &
QMaxSAT

Where are we using SAT oracles?

MaxSAT	MCS Extraction & Enumeration	MUS Extraction & Enumeration
MinSAT; Maximal Falsifiability	Lean Kernels; Backbones	MSMP
Function Minimization; Prime Enumeration; Propositional Abduction;		
Model-based Diagnosis; Axiom Pinpointing; Package Management;		
QBF & QMaxSAT	MES Extraction	

Where are we using SAT oracles?

MaxSAT

MCS Extraction
& Enumeration

MUS Extraction
& Enumeration

MinSAT; Maximal
Falsifiability

Lean Kernels;
Backbones

MSMP

Function Minimization; Prime Enumeration; Propositional Abduction;

Model-based Diagnosis; Axiom Pinpointing; Package Management;

QBF &
QMaxSAT

MES Extraction

Maximum
Cliques

Where are we using SAT oracles?

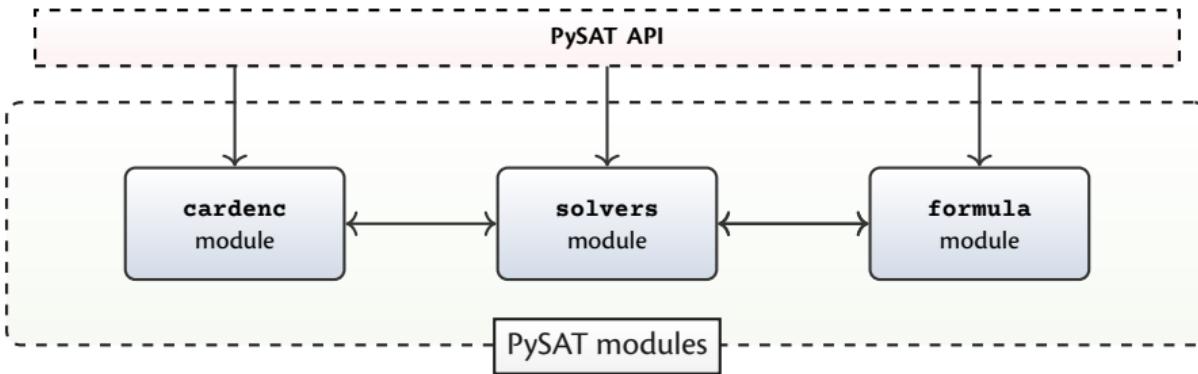
MaxSAT	MCS Extraction & Enumeration	MUS Extraction & Enumeration
MinSAT; Maximal Falsifiability	Lean Kernels; Backbones	MSMP
Function Minimization; Prime Enumeration; Propositional Abduction;		
Model-based Diagnosis; Axiom Pinpointing; Package Management;		
QBF & QMaxSAT	MES Extraction	Maximum Cliques
Explainable AI		

Some challenges

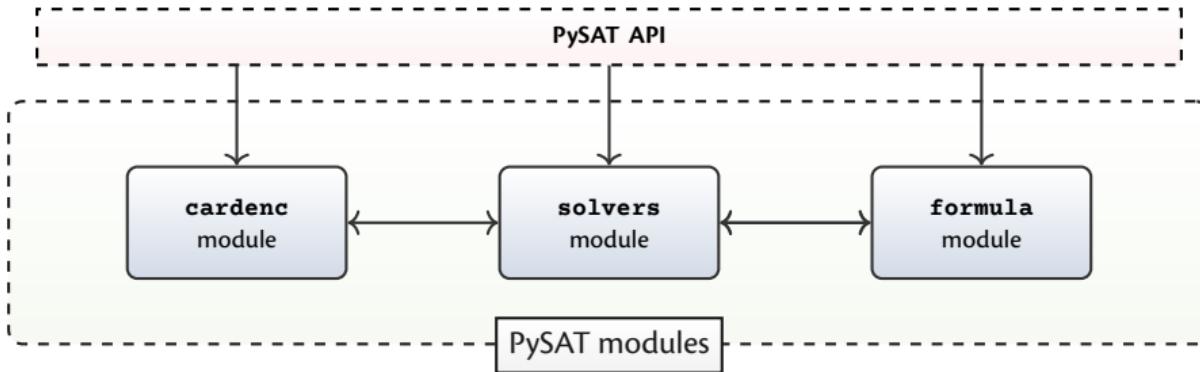
- Low-level (C/C++, even Java) implementations are **important**:
 - **Iterative** SAT solving
 - Often using **incremental** SAT
 - Need to analyze **models**
 - Need to extract **unsatisfiable cores**
 - **Many practical successes**
- But low-level implementations can be **problematic**:
 - Development time
 - Error prone
 - Difficult to maintain & change
 - ...

PySAT

Overview of PySAT

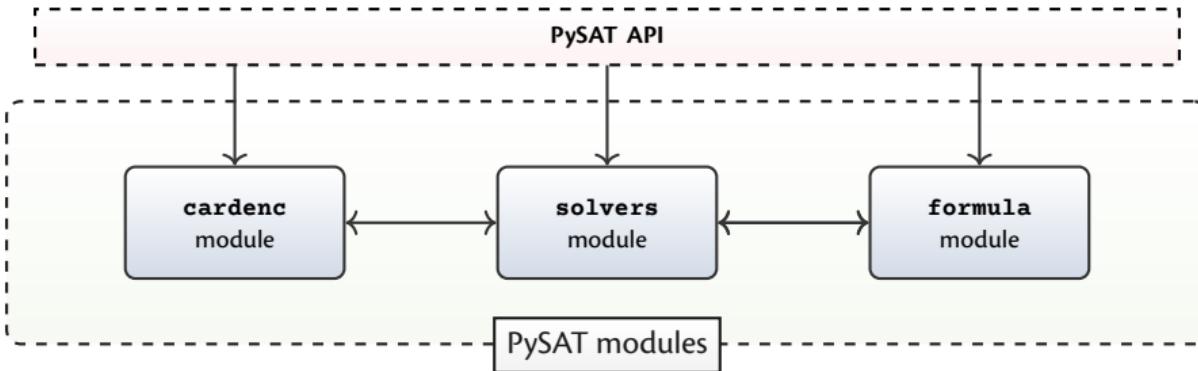


Overview of PySAT



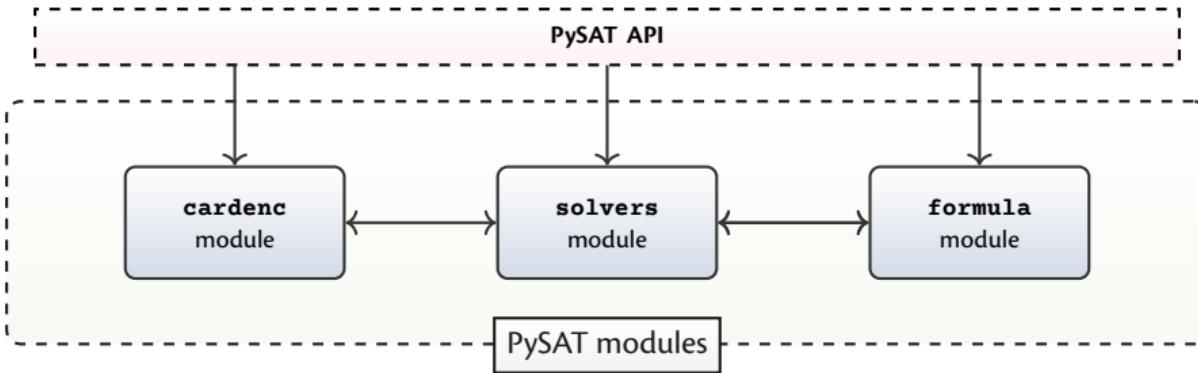
- Open source, available on [github](#)

Overview of PySAT



- Open source, available on [github](#)
- Comprehensive list of [SAT solvers](#)
- Comprehensive list of [cardinality encodings](#)
- Fairly [comprehensive documentation](#)
- A number of [examples](#)

Overview of PySAT



- Open source, available on [github](#)
- Comprehensive list of [SAT solvers](#)
- Comprehensive list of [cardinality encodings](#)
- Fairly [comprehensive documentation](#)
- A number of [examples](#)
- Optional (external) [pb](#) module

Formula manipulation (pysat.formula module)

Features

CNF & Weighted CNF (WCNF)

Read formulas from file/string

Write formulas to file

Append clauses to formula

Negate CNF formulas

Translate between CNF and WCNF

Non-clausal formula support

ID manager

- **URL:** <https://pysathq.github.io/docs/html/api/formula.html>

pysat.formula module

- **a variable in PySAT is a number from $\mathbb{N}_{>0}$**
 - e.g. $x_1 \triangleq 1$ while $x_5 \triangleq 5$

pysat.formula module

- **a variable in PySAT is a number from $\mathbb{N}_{>0}$**
 - e.g. $x_1 \triangleq 1$ while $x_5 \triangleq 5$
- **a literal is an integer**
 - e.g. $\neg x_1 \triangleq -1$ while $x_5 \triangleq 5$

pysat.formula module

- a variable in PySAT is a number from $\mathbb{N}_{>0}$
 - e.g. $x_1 \triangleq 1$ while $x_5 \triangleq 5$
- a literal is an integer
 - e.g. $\neg x_1 \triangleq -1$ while $x_5 \triangleq 5$
- a clause is a *list of literals*
 - clause $(\neg x_3 \vee x_2) \triangleq [-3, 2]$

pysat.formula module

- a variable in PySAT is a number from $\mathbb{N}_{>0}$
 - e.g. $x_1 \triangleq 1$ while $x_5 \triangleq 5$
- a literal is an integer
 - e.g. $\neg x_1 \triangleq -1$ while $x_5 \triangleq 5$
- a clause is a *list of literals*
 - clause $(\neg x_3 \vee x_2) \triangleq [-3, 2]$
- a CNF formula is an object of class CNF
 - i.e. basically it is a *list of clauses*

pysat.formula module

- a variable in PySAT is a number from $\mathbb{N}_{>0}$
 - e.g. $x_1 \triangleq 1$ while $x_5 \triangleq 5$
- a literal is an integer
 - e.g. $\neg x_1 \triangleq -1$ while $x_5 \triangleq 5$
- a clause is a *list of literals*
 - clause $(\neg x_3 \vee x_2) \triangleq [-3, 2]$
- a CNF formula is an object of class CNF
 - i.e. basically it is a *list of clauses*
- IDPool is a variable ID manager

```
1  >>> from pysat.formula import IDPool
2  >>> vpool = IDPool(occupied=[[12, 18], [3, 10]])
3  >>>
4  >>> # creating 5 unique variables for the following strings ('v1', 'v2', ..., 'v5')
5  >>> for i in range(5):
6  ...     print(vpool.id('v{0}'.format(i + 1)))
7  1
8  2
9  11
10 19
11 20
```

pysat.formula.CNF basic example

```
1  >>> from pysat.formula import CNF
2
3  >>> cnf = CNF(from_clauses=[[-1, 2], [3]])
4  >>> cnf.append([-3, 4])
5
6  >>> print(cnf.clauses)
7  [[-1, 2], [3], [-3, 4]]
8
9  >>> neg = cnf.negate()
10 >>> print(neg.clauses)
11 [[1, -5], [-2, -5], [-1, 2, 5], [3, -6], [-4, -6], [-3, 4, 6], [5, -3, 6]]
12
13 >>> print(neg.auxvars)
14 [5, -3, 6]
```

pysat.formula.CNF basic example

```
1  >>> from pysat.formula import CNF
2
3  >>> cnf = CNF(from_clauses=[[-1, 2], [3]])
4  >>> cnf.append([-3, 4])
5
6  >>> print(cnf.clauses)
7  [[-1, 2], [3], [-3, 4]]
8
9  >>> neg = cnf.negate()
10 >>> print(neg.clauses)
11 [[1, -5], [-2, -5], [-1, 2, 5], [3, -6], [-4, -6], [-3, 4, 6], [5, -3, 6]]
12
13 >>> print(neg.auxvars)
14 [5, -3, 6]
```

- + **WCNF** and **WCNFplus** formulas!

pysat.formula.CNF basic example

```
1  >>> from pysat.formula import CNF
2
3  >>> cnf = CNF(from_clauses=[[-1, 2], [3]])
4  >>> cnf.append([-3, 4])
5
6  >>> print(cnf.clauses)
7  [[-1, 2], [3], [-3, 4]]
8
9  >>> neg = cnf.negate()
10 >>> print(neg.clauses)
11 [[1, -5], [-2, -5], [-1, 2, 5], [3, -6], [-4, -6], [-3, 4, 6], [5, -3, 6]]
12
13 >>> print(neg.auxvars)
14 [5, -3, 6]
```

- + **WCNF** and **WCNFPlus** formulas!

- **much more functionality!**

(<https://pysathq.github.io/docs/html/api/formula.html>)

pysat.formula.CNF example (encoding PHP)

$$\text{PHP}_m \triangleq \bigwedge_{i=1}^{m+1} \text{AtLeast1}(x_{i1}, \dots, x_{im}) \wedge \bigwedge_{j=1}^m \text{AtMost1}(x_{1j}, \dots, x_{m+1,j})$$

$x_{ij} = \text{true} \Leftrightarrow \text{pigeon } i \text{ is at hole } j$

pysat.formula.CNF example (encoding PHP)

$$\begin{aligned} x_{ij} = \text{true} \Leftrightarrow \text{pigeon } i \text{ is at hole } j \\ \text{PHP}_m \triangleq \bigwedge_{i=1}^{m+1} \text{AtLeast1}(x_{i1}, \dots, x_{im}) \wedge \bigwedge_{j=1}^m \text{AtMost1}(x_{1j}, \dots, x_{m+1,j}) \end{aligned}$$

```
1 import itertools
2 from pysat.formula import CNF, IDPool
3
4 def PHP(nof_holes):
5     cnf = CNF()
6
7     vpool = IDPool() # initializing the pool of variable ids
8     x = lambda i, j: vpool.id('x_{0}_{1}'.format(i, j))
9
10    holes = list(range(1, nof_holes + 1)) # 1 .. m
11    pigeons = list(range(1, nof_holes + 2)) # 1 .. m + 1
12
13    for i in pigeons: # at least one hole for each pigeon
14        cnf.append([x(i, j) for j in range(1, nof_holes + 1)])
15
16    for j in holes: # at most one pigeon in a hole
17        for comb in itertools.combinations(pigeons, 2):
18            cnf.append([-x(i, j) for i in comb])
19
20    return cnf
```

Non-clausal formulas

- requires `py-aiger` package

Non-clausal formulas

- requires `py-aiger` package

```
1  >>> import aiger
2  >>> x, y, z = aiger.atom('x'), aiger.atom('y'), aiger.atom('z')
3  >>> expr = ~(x | y) & z
4  >>> print(expr.aig)
5  aag 5 3 0 1 2
6  2
7  4
8  8
9  10
10 6 3 5
11 10 6 8
12 i0 y
13 i1 x
14 i2 z
15 o0 6c454aea-c9e1-11e9-bbe3-3af9d34370a9
16
17 >>>
18 >>> from pysat.formula import CNF
19 >>> cnf = CNF(from_aiger=expr.aig)
20 >>> print(cnf.clauses)
21 [[3, 2, 4], [-3, -4], [-2, -4], [-4, -1, 5], [4, -5], [1, -5]]
22 >>> print([''{0} <-> {1}'.format(v, cnf.vpool.obj(v)) for v in cnf.inps])
23 ['3 <-> y', '2 <-> x', '1 <-> z']
24 >>> print([''{0} <-> {1}'.format(v, cnf.vpool.obj(v)) for v in cnf.outs])
25 ['5 <-> 6c454aea-c9e1-11e9-bbe3-3af9d34370a9']
```

Available solvers (pysat.solvers module)

Solver	Version
Glucose	3.0
Glucose	4.1
Lingeling	bbc-9230380-160707
Minicard	1.2
Minisat	2.2 release
Minisat	GitHub version
MapleCM	SAT competition 2018
Maplesat	MapleCOMSPS_LRB
...	...

Available solvers (`pysat.solvers` module)

Solver	Version
Glucose	3.0
Glucose	4.1
Lingeling	bbc-9230380-160707
Minicard	1.2
Minisat	2.2 release
Minisat	GitHub version
MapleCM	SAT competition 2018
Maplesat	MapleCOMSPS_LRB
...	...

- Solvers can either be used **incrementally** or **non-incrementally**

Available solvers (`pysat.solvers` module)

Solver	Version
Glucose	3.0
Glucose	4.1
Lingeling	bbc-9230380-160707
Minicard	1.2
Minisat	2.2 release
Minisat	GitHub version
MapleCM	SAT competition 2018
Maplesat	MapleCOMSPS_LRB
...	...

- Solvers can either be used [incrementally](#) or [non-incrementally](#)
- Tools can use [multiple solvers](#), e.g. for [hitting set dualization](#) or [CEGAR-based QBF solving](#)

Available solvers (`pysat.solvers` module)

Solver	Version
Glucose	3.0
Glucose	4.1
Lingeling	bbc-9230380-160707
Minicard	1.2
Minisat	2.2 release
Minisat	GitHub version
MapleCM	SAT competition 2018
Maplesat	MapleCOMSPS_LRB
...	...

- Solvers can either be used **incrementally** or **non-incrementally**
- Tools can use **multiple solvers**, e.g. for **hitting set dualization** or **CEGAR-based QBF solving**
- **Portfolio**-like use of SAT oracles is possible

Available solvers (`pysat.solvers` module)

Solver	Version
Glucose	3.0
Glucose	4.1
Lingeling	bbc-9230380-160707
Minicard	1.2
Minisat	2.2 release
Minisat	GitHub version
MapleCM	SAT competition 2018
Maplesat	MapleCOMSPS_LRB
...	...

- Solvers can either be used **incrementally** or **non-incrementally**
- Tools can use **multiple solvers**, e.g. for **hitting set dualization** or **CEGAR-based QBF solving**
- **Portfolio**-like use of SAT oracles is possible
- **URL:** <https://pysathq.github.io/docs/html/api/solvers.html>

pysat.solvers module (basic interface)

- `s = Solver() & s.delete()`

pysat.solvers module (basic interface)

- `s = Solver() & s.delete()`
- `with Solver() as s: environment`

pysat.solvers module (basic interface)

- `s = Solver() & s.delete()`
- `with Solver() as s: environment`
- `s.solve() & s.solve_limited()`

pysat.solvers module (basic interface)

- `s = Solver() & s.delete()`
- `with Solver() as s: environment`
- `s.solve() & s.solve_limited()`
- `s.add_clause() & s.append_formula()`

pysat.solvers module (basic interface)

- `s = Solver() & s.delete()`
- `with Solver() as s: environment`
- `s.solve() & s.solve_limited()`
- `s.add_clause() & s.append_formula()`
- `s.get_model() & s.get_core() & get_proof()`

pysat.solvers module (basic interface)

- `s = Solver() & s.delete()`
- `with Solver() as s: environment`
- `s.solve() & s.solve_limited()`
- `s.add_clause() & s.append_formula()`
- `s.get_model() & s.get_core() & get_proof()`

```
1  >>> from pysat.solvers import Solver, Minisat22
2
3  >>>
4  >>> s = Solver(name='g4') # accessing Glucose4 by name
5  >>> s.add_clause([-1, 2])
6  >>> s.add_clause([-1, -2])
7  >>> s.solve()
8  True
9  >>> print(s.get_model())
10 [-1, -2]
11 >>> s.delete()
12 >>>
13 >>> with Minisat22(bootstrap_with=[[-1, 2], [-1, -2]]) as m:
14 ...     print(m.solve())
15 ...     print(m.get_model())
16 True
17 [-1, -2]
```

- SAT solver is used as a [witness producing oracle](#)

- SAT solver is used as a witness producing oracle
 - satisfiable formula  return a model

- SAT solver is used as a witness producing oracle
 - **satisfiable formula**  return a **model**
 - **unsatisfiable formula**  return an **unsatisfiable core**

- SAT solver is used as a witness producing oracle
 - satisfiable formula  return a model
 - unsatisfiable formula  return an unsatisfiable core

```
1  >>> from pysat.formula import CNF, IDPool
2  >>> from pysat.solvers import Solver
3
4  >>> vpool = IDPool()
5  >>> v = lambda obj: vpool.id(obj)
6
7  >>> cnf = CNF()
8  >>> cnf.extend([[-v('a'), v('b')], [-v('a'), v('c')]])
9  >>> cnf.extend([[v('a'), -v('s1')], [-v('b'), -v('c'), -v('s2')], [v('a'), -v('c'), -v('s3')], [v('a'), -v('b'), -v('s4')]])
10
11 >>> with Solver(bootstrap_with=cnf) as s:
12 ...     if s.solve(assumptions=[v('s1'), -v('s2'), -v('s3'), v('s4')]) == True:
13 ...         print('model:', ['{0}{1}'.format(' if v > 0 else '-' , vpool.obj(abs(v))) for v in s.get_model()])
14 ...
15 ...     if s.solve(assumptions=[v('s1'), v('s2'), v('s3'), v('s4')]) == False:
16 ...         print('ucore:', ['{0}{1}'.format(' if v > 0 else '-' , vpool.obj(abs(v))) for v in s.get_core()])
17 ...
18 model: ['a', 'b', 'c', 's1', '-s2', '-s3', 's4']
19 ucore: ['s2', 's1']
```

pysat.solvers — solve_limited()

```
1  >>> from pysat.solvers import MinisatGH
2  >>> from pysat.examples.genhard import Parity
3  >>>
4  >>> cnf = Parity(size=10) # too hard for a SAT solver
5  >>> m = MinisatGH(bootstrap_with=cnf.clauses)
6  >>>
7  >>> m.prop_budget(100000) # doing at most 100000 propagations
8  >>> print(m.solve_limited()) # making a limited oracle call
9  None
10 >>> m.delete()
```

pysat.solvers — solve_limited() and interrupt()

```
1  >>> from pysat.solvers import MinisatGH
2  >>> from pysat.examples.genhard import PHP
3  >>> from threading import Timer
4  >>>
5  >>> cnf = PHP(nof_holes=20) # PHP20 is too hard for a SAT solver
6  >>> m = MinisatGH(bootstrap_with=cnf)
7  >>>
8  >>> def interrupt(s):
9  >>>     s.interrupt()
10 >>>
11 >>> timer = Timer(10, interrupt, [m])
12 >>> timer.start()
13 >>>
14 >>> print(m.solve_limited(expect_interrupt=True))
15 None
16 >>> m.delete()
```

pysat.solvers — propagate()

```
1  >>> from pysat.solvers import Glucose3
2  >>> from pysat.card import *
3  >>>
4  >>> cnf = CardEnc.atmost(lits=range(1, 6), bound=1, encoding=EncType.pairwise)
5  >>> g = Glucose3(bootstrap_with=cnf.clauses)
6  >>>
7  >>> g.propagate(assumptions=[1])
8  (True, [1, -2, -3, -4, -5])
9  >>>
10 >>> g.add_clause([2])
11 >>> g.propagate(assumptions=[1])
12 (False, [])
13 >>>
14 >>> g.delete()
```

- + lots of other cool stuff!

<https://pysathq.github.io/docs/html/api/solvers.html>

Available cardinality encodings (pysat.card module)

Name	Type
pairwise	AtMost1
bitwise	AtMost1
ladder	AtMost1
sequential counter	AtMostk
sorting network	AtMostk
cardinality network	AtMostk
totalizer	AtMostk
mtotalizer	AtMostk
kmtotalizer	AtMostk

- Also **AtLeastK** and **EqualsK** constraints
- **URL:** <https://pysathq.github.io/docs/html/api/card.html>

pysat.card module

- `atleast()`, `atmost()`, and `equals()`

pysat.card module

- `atleast()`, `atmost()`, and `equals()`
- `ITotalizer` — *iterative totalizer*

pysat.card module

- `atleast()`, `atmost()`, and `equals()`
- `ITotalizer` — *iterative totalizer*

```
1  >>> from pysat.card import EncType, CardEnc
2
3  >>> cnf = CardEnc.atmost(lits=[1, 2, 3], bound=1, encoding=EncType.pairwise)
4  >>> print(cnf.clauses)
5  [[-1, -2], [-1, -3], [-2, -3]]
6
7  >>> cnf = CardEnc.equals(lits=[1, 2, 3], bound=1, encoding=EncType.pairwise)
8  >>> print(cnf.clauses)
9  [[1, 2, 3], [-1, -2], [-1, -3], [-2, -3]]
```

pysat.card — ITotalizer

- encode $\sum x_i \leq K$ constraints

pysat.card — ITotalizer

- encode $\sum x_i \leq K$ constraints
- extend `left`-hand side with `extend()`

pysat.card — ITotalizer

- encode $\sum x_i \leq K$ constraints
- extend **left**-hand side with `extend()`
- increase **right**-hand side with `increase()`

pysat.card — ITotalizer

- encode $\sum x_i \leq K$ constraints
- extend **left**-hand side with `extend()`
- increase **right**-hand side with `increase()`
- *merge* with another totalizer, with `merge_with()`

pysat.card — ITotalizer

- encode $\sum x_i \leq K$ constraints
- extend **left**-hand side with `extend()`
- increase **right**-hand side with `increase()`
- *merge* with another totalizer, with `merge_with()`
- set bound **incrementally**

pysat.card — ITotalizer

- encode $\sum x_i \leq K$ constraints
- extend left-hand side with `extend()`
- increase right-hand side with `increase()`
- merge with another totalizer, with `merge_with()`
- set bound **incrementally**

```
1  >>> from pysat.card import ITotalizer
2
3  >>> t = ITotalizer(lits=[1, 2], ubound=1)
4  >>> print(t.cnf.clauses)
5  [[-2, 3], [-1, 3], [-1, -2, 4]]
6  >>> print(t.rhs)
7  [3, 4]
8  >>>
9  >>> t.extend(lits=[5], ubound=2) # adding one more literal and increasing the bound
10 >>> print(t.cnf.clauses)
11 [[-2, 3], [-1, 3], [-1, -2, 4], [-5, 6], [-3, 6], [-4, 7], [-3, -5, 7], [-4, -5, 8]]
12 >>>
13 >>> print(t.cnf.clauses[-t.nof_new:]) # these are newly added clauses
14 [[-5, 6], [-3, 6], [-4, 7], [-3, -5, 7], [-4, -5, 8]]
15 >>>
16 >>> print(t.rhs)
17 [6, 7, 8] # these literals represent bounds 0, 1, and 2
18 >>> t.delete()
```

More examples — `pysat.examples` module

- `models.py` — simple model enumerator

More examples — `pysat.examples` module

- `models.py` — simple model enumerator
- `genhard.py` — a few hard combinatorial principles

More examples — `pysat.examples` module

- `models.py` — simple model enumerator
- `genhard.py` — a few hard combinatorial principles
- `hitman.py` — minimal hitting set enumerator

More examples — `pysat.examples` module

- `models.py` — simple model enumerator
- `genhard.py` — a few hard combinatorial principles
- `hitman.py` — minimal hitting set enumerator
- `musx.py` — deletion-base MUS extractor

More examples — `pysat.examples` module

- `models.py` — simple model enumerator
- `genhard.py` — a few hard combinatorial principles
- `hitman.py` — minimal hitting set enumerator
- `musx.py` — deletion-base MUS extractor
- `optux.py` — smallest MUS extractor

More examples — `pysat.examples` module

- `models.py` — simple model enumerator
- `genhard.py` — a few hard combinatorial principles
- `hitman.py` — minimal hitting set enumerator
- `musx.py` — deletion-base MUS extractor
- `optux.py` — smallest MUS extractor
- `lbx.py` — LBX-like MCS enumeration

More examples — `pysat.examples` module

- `models.py` — simple model enumerator
- `genhard.py` — a few hard combinatorial principles
- `hitman.py` — minimal hitting set enumerator
- `musx.py` — deletion-base MUS extractor
- `optux.py` — smallest MUS extractor
- `lbx.py` — LBX-like MCS enumeration
- `mcls.py` — CLD-like MCS enumeration

More examples — `pysat.examples` module

- `models.py` — simple model enumerator
- `genhard.py` — a few hard combinatorial principles
- `hitman.py` — minimal hitting set enumerator
- `musx.py` — deletion-base MUS extractor
- `optux.py` — smallest MUS extractor
- `lbx.py` — LBX-like MCS enumeration
- `mcls.py` — CLD-like MCS enumeration
- `fm.py` — Fu&Malik MaxSAT algorithm

More examples — `pysat.examples` module

- `models.py` — simple model enumerator
- `genhard.py` — a few hard combinatorial principles
- `hitman.py` — minimal hitting set enumerator
- `musx.py` — deletion-base MUS extractor
- `optux.py` — smallest MUS extractor
- `lbx.py` — LBX-like MCS enumeration
- `mcls.py` — CLD-like MCS enumeration
- `fm.py` — Fu&Malik MaxSAT algorithm
- `lsu.py` — LSU MaxSAT algorithm

More examples — `pysat.examples` module

- `models.py` — simple model enumerator
- `genhard.py` — a few hard combinatorial principles
- `hitman.py` — minimal hitting set enumerator
- `musx.py` — deletion-base MUS extractor
- `optux.py` — smallest MUS extractor
- `lbx.py` — LBX-like MCS enumeration
- `mcls.py` — CLD-like MCS enumeration
- `fm.py` — Fu&Malik MaxSAT algorithm
- `lsu.py` — LSU MaxSAT algorithm
- **`rc2.py` — award-winning MaxSAT solver**



Encoding Sudoku

```
1  def encode_sudoku():
2      cnf = CNF() # this is where we store the result formula
3
4      pool = IDPool() # pool of variable ids
5      var = lambda i, j, v: pool.id(tuple([i + 1, j + 1, v + 1]))
6
7      # at least one value in each cell
8      for i, j in itertools.product(range(9), range(9)):
9          cnf.append([var(i, j, val) for val in range(9)])
10
11     # at most one value in each row
12     for i in range(9):
13         for val in range(9):
14             for j1, j2 in itertools.combinations(range(9), 2):
15                 cnf.append([-var(i, j1, val), -var(i, j2, val)])
16
17     # at most one value in each column
18     for j in range(9):
19         for val in range(9):
20             for i1, i2 in itertools.combinations(range(9), 2):
21                 cnf.append([-var(i1, j, val), -var(i2, j, val)])
22
23     # at most one value in each square
24     for val in range(9):
25         for i in range(3):
26             for j in range(3):
27                 subgrid = itertools.product(range(3 * i, 3 * i + 3), range(3 * j, 3 * j + 3))
28                 for c in itertools.combinations(subgrid, 2):
29                     cnf.append([-var(c[0][0], c[0][1], val), -var(c[1][0], c[1][1], val)])
30
31     return cnf
```

Interaction with plenty of other libraries in Python!

- **Sudoku demo** — pygtk
- **MAPF demo** — mcpi.minecraft

Interaction with plenty of other libraries in Python!

- **Sudoku demo** — pygtk
- **MAPF demo** — mcpi.minecraft
- **math and data** — numpy, scipy, pandas, gurobi,...

Interaction with plenty of other libraries in Python!

- **Sudoku demo** — pygtk
- **MAPF demo** — mcpi.minecraft
- **math and data** — numpy, scipy, pandas, gurobi,...
- **XAI** — scikit, pytorch, tensorflow,...
- ...

Interaction with plenty of other libraries in Python!

- **Sudoku demo** — pygtk
- **MAPF demo** — mcpi.minecraft
- **math and data** — numpy, scipy, pandas, gurobi,...
- **XAI** — scikit, pytorch, tensorflow,...
- ...
- **DL** — ?

Installation & platforms

- `$ pip install python-sat[pplib,aiger]`

Installation & platforms

- `$ pip install python-sat[pplib,aiger]`
- **Linux, MacOS, Windows**

Installation & platforms

- `$ pip install python-sat[pplib,aiger]`
- **Linux, MacOS, Windows**
- **no compilation needed**

Installation & platforms

- `$ pip install python-sat[pplib,aiger]`
- **Linux, MacOS, Windows**
- **no compilation needed**
- + **online REPL platforms, e.g. repl.it**

- `$ pip install python-sat[plib,aiger]`
- **Linux, MacOS, Windows**
- **no compilation needed**
- + **online REPL platforms, e.g. repl.it**
- + **PySAT is a part of Pyodide**
(Python scientific stack compiled to WebAssembly)



<https://pysathq.github.io/>