OptiLog: A Framework for SAT-based Systems

The 24th International Conference on Theory and Applications of Satisfiability Testing

Carlos Ansótegui Jesús Ojeda Antonio Pacheco
Josep Pon Josep M. Salvia Eduard Torres

Logic & Optimization Group (LOG), University of Lleida, Spain

Barcelona - July 7, 2021

ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA

- Rapid prototyping of SAT-based systems using Python as programming language.
- Uniform interface for SAT solvers and Pseudo-Boolean (PB) encoders.
- Simple methodology to integrate any C/C++ SAT solver.
- Framework to simplify the Automatic Configuration (AC) process of SAT-based systems.

- PySAT [IMM18] is a Python toolkit that integrates a number of widely used state-of-the-art SAT solvers, as well as a variety of cardinality and Pseudo-Boolean encodings.
- IPASIR [Bc14] is a simple C interface to incremental SAT solvers. This interface is used in the SAT competition's incremental track.
- The PBLib [PS15] is a C++ project that collects and implements many different encodings for Pseudo-Boolean constraints into conjunctive normal form (CNF).

OptiLog[1] is a unified framework for developing SAT-based systems.

It provides a new way of integrating SAT solvers into Python without implementing Python code or Python/C API bindings, a unified interface for encoding PB constraints and a framework that simplifies the automatic configuration of any Python function.

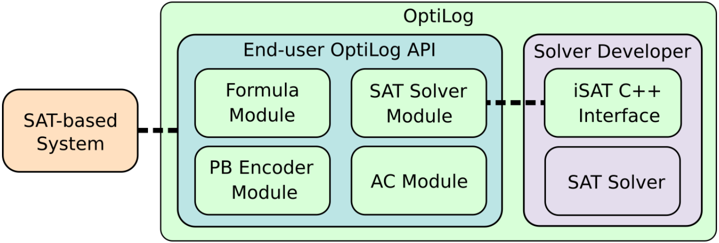
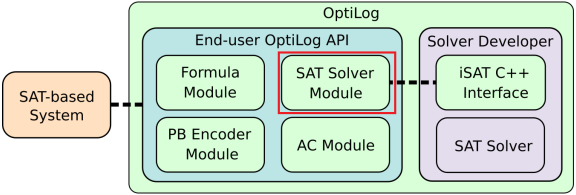


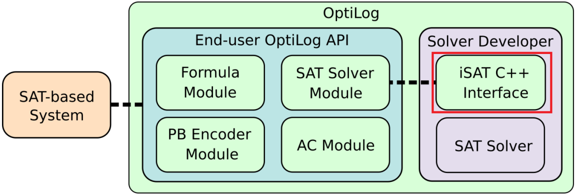
Figure: OptiLog Architecture

[1] Docs: http://ulog.udl.cat/static/doc/optilog/html/index.html




```
>>> from optilog.loaders import load_cnf
>>> from optilog.sat import Glucose41
>>> solver = Glucose41()
>>> load_cnf('path/to/formula.cnf', solver)
>>> solver.solve()
True
>>> solver.model()
[1, -2, 3, 4, ...]
```

OptiLog allows the integration of SAT solvers written in C/C++ to Python through the implementation of the iSAT interface.

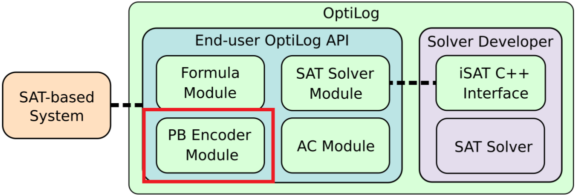



```
class Glucose41Wrapper : iSAT
{
    int newVar();
    void addClause(vector<int> &literals);
    E_STATE solve(vector<int> &assumptions);
    void getModel(vector<int> &model);
    (...)
}
```

The iSAT interface provides these new additional methods:

- `set_decision_var`
- `set_static_heuristic`
- `solve_hard_limited`
- `learnt_clauses`
- get and set methods for solver configuration

Integration of PBLib encodings and the *Totalizer* encoding from PySAT using the same interface.



Pseudo-Boolean encodings:

- Adder networks
- Binary merge
- BDD
- Sequential weigh counters
- Sorting networks

Cardinality encodings:

- Bitwise
- Cardinality network
- Ladder / Regular
- Pairwise
- Sequential counters
- Totalizer

- Non-incremental encoders

Example: $2x + y + z \geq 2$

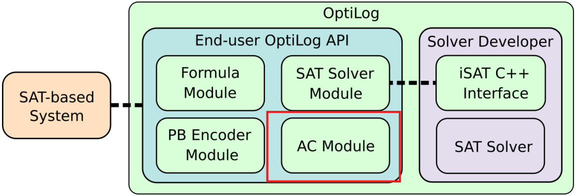
```
>>> from optilog.sat.pbencoder import Encoder
>>> Encoder.at_least_k([1,2,3], 2, [2,1,1], "bdd")
(9, [[4], [-4, 5], [3, 5], [-4, 3, 6], ...])
```

- Incremental encoders

Example: From $x + y + z \leq 3$ to $x + y + z \leq 2$

```
>>> from optilog.sat.pbencoder import IncrementalEncoder
>>> enc, max_var, C = IncrementalEncoder.init([1,2,3], 3)
(<IncrementalEncoder object>, 6, [[-3, 4], ...])
>>> max_var, C_ext = enc.extend(2)
(6, [[-6]])
```

The Automatic Configuration (AC) module provides an API to automatically generate the configuration files that automatic configurators need as input to configure any Python function.



OptiLog currently provides support for SMAC [HHL11] and GGA [AST09, AMS^{+}15].

```python
# example.py

def algorithm(instance, seed):
    s = Glucose41()
    s.set("seed", seed)
    f = load_cnf(instance, s)
    max_var = f.max_var()
    _, C = Encoder.at_most_k(
        range(1, max_var + 1),
        bound=max_var // 2,
        max_var=max_var,
    )
    s.add_clauses(C)
    res = s.solve()
    print("s", res)
    return res
```

```python
# example.py

def algorithm(instance, seed):
    s = Glucose41()
    s.set("seed", seed)
    f = load_cnf(instance, s)
    max_var = f.max_var()
    _, C = Encoder.at_most_k(
        range(1, max_var + 1),
        bound=max_var // 2,
        max_var=max_var,
    )
    s.add_clauses(C)
    res = s.solve()
    print("s", res)
    return res
```

```python
# example_ac.py

from optilog.loaders import load_cnf
from optilog.sat.pbencoder import Encoder

from optilog.autocfg import ac, Categorical
from optilog.autocfg import CfgCall
from optilog.autocfg.sat import get_glucose41

ENCODERS = ["best", "cardnetwrk", "totalizer"]

@ac
def algorithm(
    instance, seed,
    init_solver_fn: CfgCall(get_glucose41),
    encoding: Categorical(*ENCODERS) = "best",
):
    s = init_solver_fn(seed=seed)
    f = load_cnf(instance, s)
    max_var = f.max_var()
    _, C = Encoder.at_most_k(
        range(1, max_var + 1),
        bound=max_var // 2,
        max_var=max_var,
        encoding=encoding,
    )
    s.add_clauses(C)
    res = s.solve()
    print("s", res)
    return res
```

Example: Automatically configuring a SAT-based algorithm (II)

- GGA scenario generation

```python
# scenario_gga.py

from pathlib import Path
from optilog.autocfg.configurators import GGAConfigurator
from example_ac import algorithm

if __name__ == "__main__":
    configurator = GGAConfigurator(
        algorithm, runsolver_path="./runsolver",
        global_cfgcalls=[algorithm],
        input_data=["inst1.cnf", ..., "instN.cnf"],
        data_kwarg="instance", seed_kwarg="seed",
        eval_time_limit=30, memory_limit=4 * 1024,
        tuner_rt_limit=180, run_obj="runtime",
        quality_regex=r"^s (?:True|False)$",
        seed=1, cost_min=0, cost_max=300,
    )
    configurator.generate_scenario("./scenario-gga")
```

- GGA execution

```
./pydgga gga -s scenario-gga/ --slots 1
```

Example: Automatically configuring a SAT-based algorithm (II)

- GGA scenario generation

```python
# scenario_gga.py

from pathlib import Path
from optilog.autocfg.configurators import GGAConfigurator
from example_ac import algorithm

if __name__ == "__main__":
    configurator = GGAConfigurator(
        algorithm, runsolver_path="./runsolver",
        global_cfgcalls=[algorithm],
        input_data=["inst1.cnf", ..., "instN.cnf"],
        data_kwarg="instance", seed_kwarg="seed",
        eval_time_limit=30, memory_limit=4 * 1024,
        tuner_rt_limit=180, run_obj="runtime",
        quality_regex=r"^s (?:True|False)$",
        seed=1, cost_min=0, cost_max=300,
    )
    configurator.generate_scenario("./scenario-gga")
```

- GGA execution

```
./pydgga gga -s scenario-gga/ --slots 1
```

- What about configuring with SMAC?

Example: Automatically configuring a SAT-based algorithm (III)

- SMAC scenario generation

```python
# scenario_smac.py

from optilog.autocfg import SMACConfigurator
from example_ac import algorithm

if __name__ == "__main__":
    configurator = SMACConfigurator(
        algorithm, runsolver_path="./runsolver",
        global_cfgcalls=[algorithm],
        input_data=["inst1.cnf", ..., "instN.cnf"],
        data_kwarg="instance", seed_kwarg="seed",
        cutoff=30, memory_limit=4 * 1024,
        wallclock_limit=180, run_obj="runtime",
        quality_regex=r"^s (?:True|False)$",
    )
    configurator.generate_scenario("./scenario")
```

- SMAC execution

```
smac --scenario ./scenario/scenario.txt
```

Conclusions

- OptiLog eases the access to solvers and encoders to our and other communities.
- The iSAT interface could become the basis for an standard SAT API.
- The AC module can potentially be applied to tune any Python function.

Future Work

- We will add other solvers, like MaxSAT or PB solvers.
- Support for more complex compilation pipelines.
- Support for callback functions on critical points as in Gurobi [Gur21] (restarts, pick literal decision, conflict analysis, etc.).
- Integrate crafted and random instance generators and allow dynamic instance downloading from repositories.

[AMS+15] Carlos Ansótegui, Yuri Malitsky, Horst Samulowitz, Meinolf Sellmann, and Kevin Tierney.
Model-based genetic algorithms for algorithm configuration.
In *IJCAI*, pages 733–739, 2015.

[AST09] C. Ansotegui, M. Sellmann, and K. Tierney.
A gender-based genetic algorithm for the automatic configuration of algorithms.
In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, pages 142–157, 2009.

[Bc14] Tomas Balyo and contributors.
The standard interface for incremental satisfiability solving.
https://github.com/biotomas/ipasir, 2014.

[Gur21] Gurobi Optimization.
Gurobi.
https://www.gurobi.com/, 2021.

[HHL11] F. Hutter, H.H. Hoos, and K. Leyton-Brown.
Sequential model-based optimization for general algorithm configuration.
In *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, pages 507–523, 2011.

[IMM18] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva.
PySAT: A Python toolkit for prototyping with SAT oracles.
In *SAT*, pages 428–437, 2018.

[PS15] Tobias Philipp and Peter Steinke.
PBLib – a library for encoding pseudo-boolean constraints into CNF.
In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 9–16, Cham, 2015. Springer International Publishing.