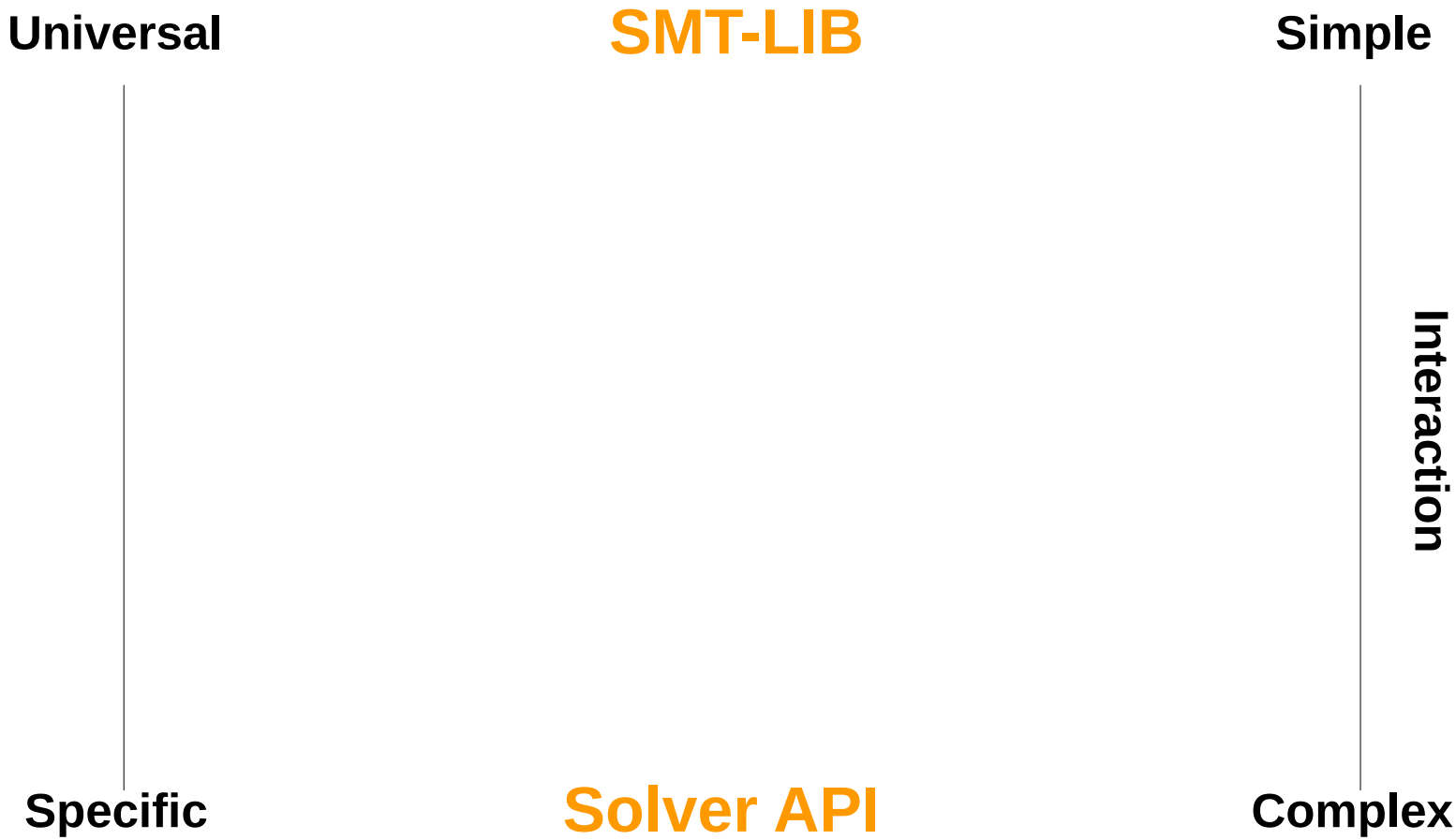
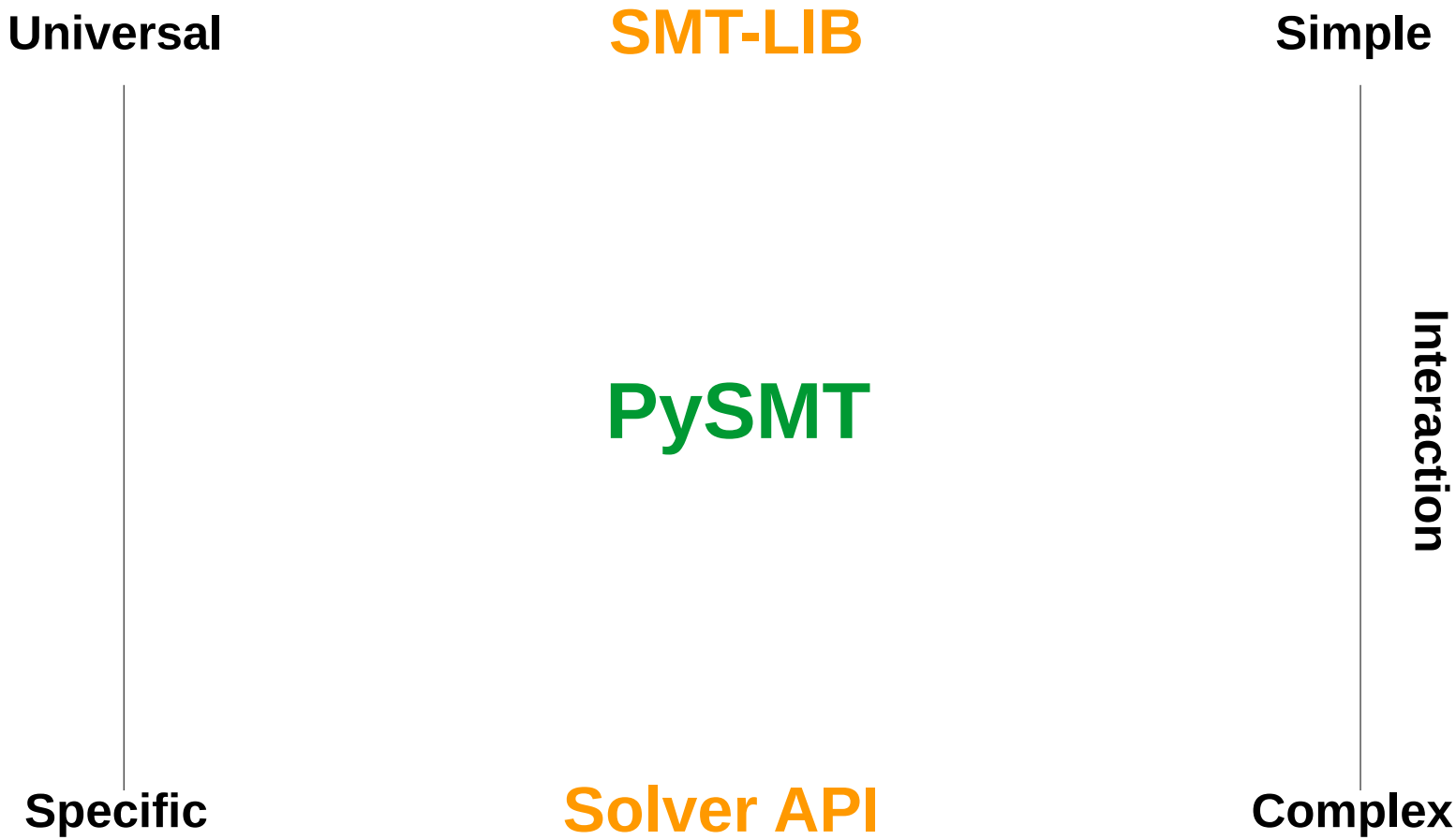
PySMT: a Solver-Agnostic Library for Fast Prototyping of SMT-Based Algorithms

Marco Gario and Andrea Micheli
gario@fbk.eu

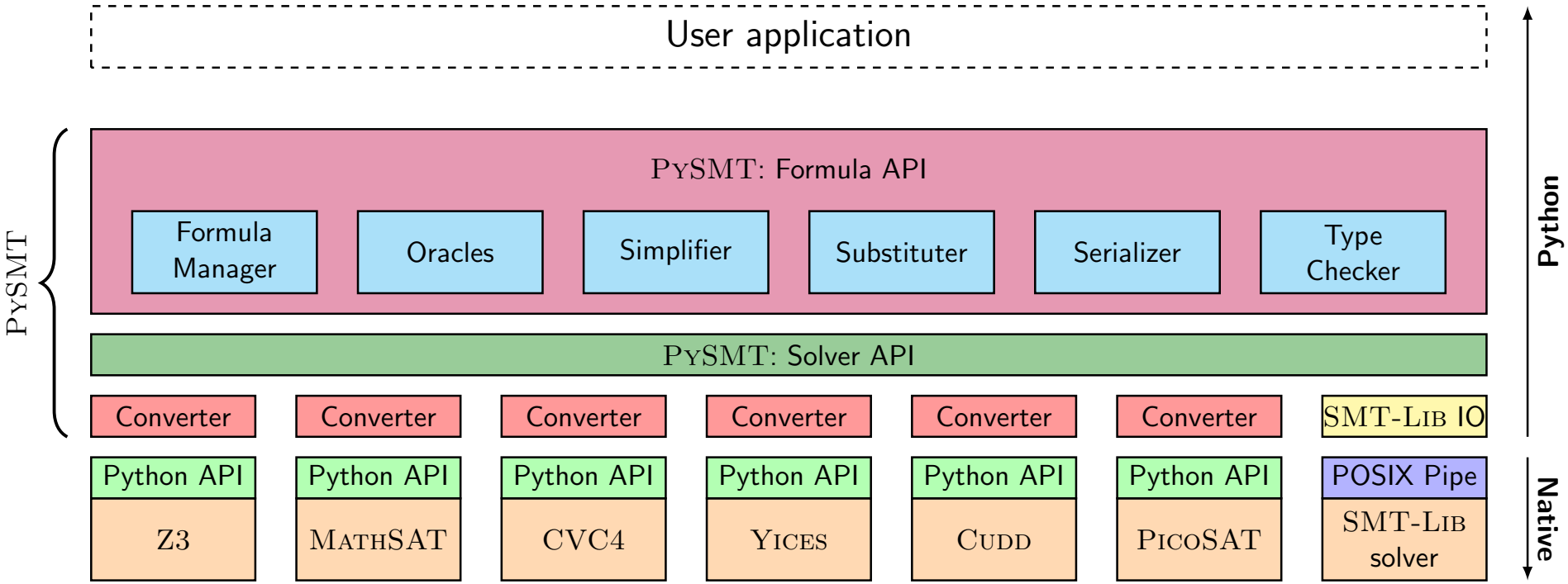
Fondazione Bruno Kessler (FBK)
University of Trento

2015-07-18


Universal
SMT-LIB
Simple
Interaction
Specific
Solver API
Complex


Universal
SMT-LIB
Simple
Interaction
PySMT
Specific
Solver API
Complex

PySMT



Simplify prototyping + Experiment with multiple solvers

$$\mathsf{H}+\mathsf{E}+\mathsf{L}+\mathsf{L}+\mathsf{O} = \mathsf{W}+\mathsf{O}+\mathsf{R}+\mathsf{L}+\mathsf{D} = 25$$

Hello World

```
1  from pysmt.shortcuts import *
2  from pysmt.typing import INT
3
4  hello = [Symbol(s, INT) for s in "hello"]
5  world = [Symbol(s, INT) for s in "world"]
6  letters = set(hello+world)
7  domains = And([And(GE(l, Int(1)),
8                     LT(l, Int(10))) for l in letters])
9
10 sum_hello = Plus(hello) # n-ary operators can take lists
11 sum_world = Plus(world) # as arguments
12 problem = And(Equals(sum_hello, sum_world),
13                Equals(sum_hello, Int(25)))
14 formula = And(domains, problem)
15
16 print("Serialization of the formula:")
17 print(formula)
18
19 model = get_model(formula, solver_name="z3") # Try msat
20
21 if model: print(model)
22 else: print("No solution found")
```

Features: Solvers and Logics

- Supported Logics: $\mathcal{UFLIRA}$ and subsets + $\mathcal{BV}$
- Solvers:
  - Z3, MathSAT 5, CVC4, Yices, PicoSAT, Cudd
  - **Any** SMT-LIB2 Solver
- Quantifier Elimination ($\mathcal{LIA}$, $\mathcal{LRA}$):
  - Z3
  - MathSAT

Quantifier Elimination

1. Build quantified expression f
2. Eliminate quantifier using Z3
3. Solve using CVC4

```
1  x, y, z = [Symbol(s, REAL) for s in "xyz"]
2
3  f = ForAll([x], (x < 5) | ((x + y + z) >= 8.0))
4
5  qf_f = qelim(f, solver_name="z3")
6
7  res = is_sat(qf_f, solver_name="cvc4")
```

EF-SMT

Problems of the form

$$\exists \vec{x}.\ \forall \vec{y}.\ \varphi(\vec{x}, \vec{y})$$

- Solve without quantifier elimination
- 2 Solvers: Existential and Universal

EF-SMT

Problems of the form

$$\exists \vec{x}.\ \forall \vec{y}.\ \varphi(\vec{x}, \vec{y})$$

- Solve without quantifier elimination
- 2 Solvers: Existential and Universal

1. Find a model τ for φ over $\vec{x}$

$\rightarrow$ Not Found: UNSAT

2. Find a model σ for $\neg\varphi[\vec{x}/\tau]$ over $\vec{y}$

$\rightarrow$ Not Found: SAT

3. Add constraint $\varphi[\vec{y}/\sigma]$

EF-SMT

```
1  with Solver(name=esolver_name) as esolver:
2      esolver.add_assertion(Bool(True))
3
4      while True:
5          eres = esolver.solve()
6          if not eres:
7              return False # UNSAT
8          # Extract model and perform substitution
9          tau = {v: esolver.get_value(v) for v in x}
10         sub_phi = phi.substitute(tau).simplify()
11
12         fmodel = get_model(Not(sub_phi),
13                            solver_name=fsolver_name)
14
15         if fmodel is None:
16             return tau # SAT (+ Model)
17         sigma = {v: fmodel[v] for v in y}
18         sub_phi = phi.substitute(sigma).simplify()
19         # Add constraint to existential part and restart
20         esolver.add_assertion(sub_phi)
```

Features Overview

- Automatic Logic detection
- Unified Model Representation
- Unsat-Core
- Interpolants
- SMT-LIB Support
- Access to solver-specific features
- Typechecking, Substitution, Printing, Simplification
- Infix Notation

Integrating your Solver

You need a low-level API to **create** and solve **expressions**

1. Wrap low-level API in Python (e.g., SWIG)
2. Implement `pysmt.solvers.solver.Converter` interface
3. Implement `pysmt.solvers.solver.Solver` interface

Thin Wrappers: directly access a given solver

```
1  import mathsat
2  from pysmt.shortcuts import Or, Symbol, Solver, And
3
4  def callback(model, converter, result):
5      py_model = [converter.back(v) for v in model]
6      result.append(And(py_model))
7      return 1 # go on
8
9  x, y = Symbol("x"), Symbol("y")
10 f = Or(x, y)
11
12 msat = Solver(name="msat")
13 converter = msat.converter
14
15 msat.add_assertion(f)
16 result = []
17 mathsat.msat_all_sat(msat.msat_env, # MathSat API call
18       [converter.convert(x)],
19       lambda model : callback(model, converter, result))
20
21 print("exists y . %s is equivalent to %s" % \
22       (f, Or(result)))
```

Case-studies

- Temporal Networks (Constraints 2015):
  - Quantifier Elimination for Temporal Uncertainty
  - Max-SAT algorithm for Strategy Construction
- TFPG Validation (AAAI'15):
  - Quantifier Elimination for Refinement Check
  - Benchmarking: Exploit Python library for random graph generation (`networkx`)

Related

- Libraries for other languages work by pipe through SMT-LIB

$\Rightarrow$ Missing functionalities: **Quantifier Elimination**

- *metaSMT*: Using C++ templates for adapting native APIs (Only BV and Array)
- *SMT-KIT*: C++ library, supports most theories (QF)
- Neither provides **unified** handling of **models** or utilities to simplify **expressions manipulation**

Future Work

- **Theories**: Arrays, Non-linear Arithmetic, ...
- **Solvers**: Boolector, OpenSMT, ... `Your Solver` ...
- Simplify **installation** of solvers

Conclusion

PySMT:

- Solver agnostic SMT
- Fast-prototyping
- Combine multiple solvers

Info and Contributing

```
$ pip install pysmt
```

or

```
$ git clone https://github.com/pysmt/pysmt
```

Examples and Tests to get started

Open-source License: APACHE v2

Feedback, **bug reports** and **contributions** are welcome!

Info and Contributing

```
$ pip install pysmt
```

or

```
$ git clone https://github.com/pysmt/pysmt
```

Examples and Tests to get started

Open-source License: APACHE v2

Feedback, **bug reports** and **contributions** are welcome!

Thank You!

Marco Gario and Andrea Micheli - gario@fbk.eu

Support Summary

Solver	pySMT name	Logics	SAT	Model	UNSAT-Core
MathSAT	`msat`	QF_UFLIRA, QF_BV	Yes	Yes	Yes
Z3	`z3`	UFLIRA, QF_BV	Yes	Yes	Yes
CVC4	`cvc4`	QF_UFLIRA, QF_BV	Yes	Yes	No
Yices	`yices`	QF_UFLIRA, QF_BV	Yes	Yes	No
SMT-Lib Interface	`<custom>`	UFLIRA, QF_BV	Yes	Yes	No [Yes]
PicoSAT	`picosat`	QF_BOOL	Yes	Yes	No [Yes]
BDD (CUDD)	`bdd`	BOOL	Yes	Yes	No

Quantifier Eliminator	pySMT name	Logics
MathSAT FM	`msat-fm`	LRA
MathSAT LW	`msat-lw`	LRA
Z3	`z3`	LRA, LIA
BDD (CUDD)	`bdd`	BOOL

Interpolator	pySMT name	Logics
MathSAT	`msat`	QF_UFLIA, QF_UFLRA, QF_BV
Z3	`z3`	QF_UFLIA, QF_UFLRA

Why Infix notation is discouraged?

1. Precedence of operators
2. Equality overloading

```
f = ForAll([x], x < 5 | (x + y + z) >= 8.0)
f = ForAll([x], (x < 5) | (x + y + z) >= 8.0)
f = ForAll([x], (x < 5) | ((x + y + z) >= 8.0))
```

BMC

- Most work goes into substitution
- Substitutions are a Map (Dictionary)

```
1  def unroll_prop(prop, k):
2      not_prop_up_to_k = []
3      vs = prop.get_free_variables()
4      for i in xrange(k):
5          renaming = {v : var_at_time(v, i) for v in vs}
6          p_i = prop.substitute(renaming)
7          not_prop_up_to_k.append(Not(p_i))
8      return Or(not_prop_up_to_k)
```