

1. Objectives

Main goal: a framework for experimenting with BDD-based model-checking algorithms.

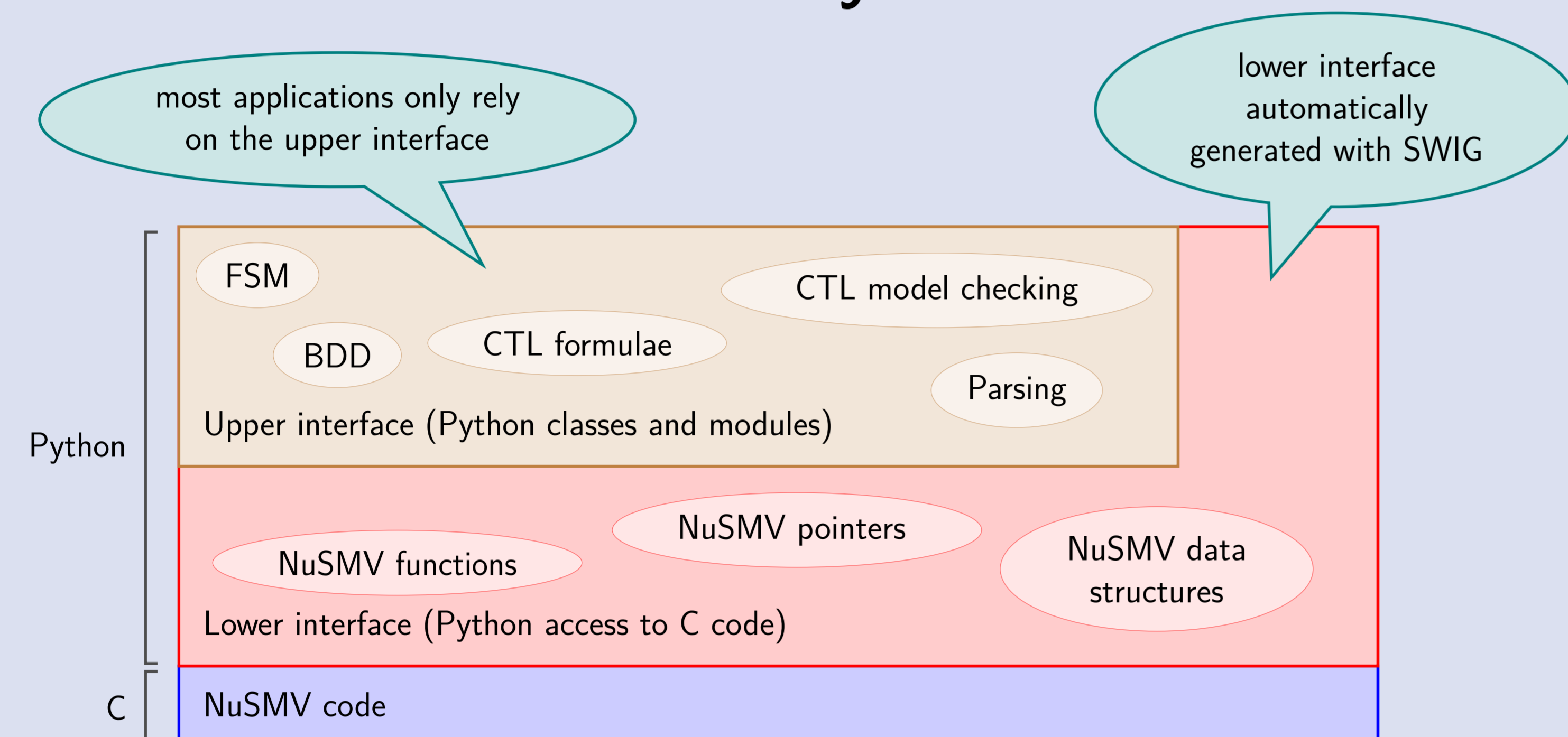
+ Access to **NuSMV features:** modeling language, BDD library, model checking, etc.

+ A **high-level language:** hide low-level mechanisms such as memory management.

⇒ **PyNuSMV:** a Python platform to implement new logics and experiment with custom model-checking algorithms based on NuSMV.

Already used for prototyping ARCTL, CTLK, ATL model checking and rich counter-examples.

2. Structure of the library



4. Example: ATL model checking

```
1 def eval_atl(fsm, spec):
2 ...
3 elif type(spec) is CAG:
4     return ~ceu(fsm,
5                 {atom.value for atom in spec.group},
6                 BDD.true(fsm.bddEnc.DDmanager),
7                 ~eval_atl(fsm, spec.child))
8 ...
```

$$\llbracket \Gamma \rrbracket G \phi = \neg \llbracket \Gamma \rrbracket \text{True } U \neg \phi$$

```
1 def ceu(fsm, agents, phi, psi):
2     return fp(lambda Y: psi |
3               (phi & fsm.pre_strat(Y, agents)),
4               BDD.false(fsm.bddEnc.DDmanager))
```

$$\langle \Gamma \rangle \phi U \psi = \mu Z. \psi \vee (\phi \wedge \text{Pre}_{\langle \Gamma \rangle}(Z))$$

```
1 def pre_strat(self, states, agents):
2     gamma_cube = self.inputs_cube_for_agents(agents)
3     ngamma_cube = self.bddEnc.inputsCube - gamma_cube
4     return (~self.weak_pre(~states).forsome(ngamma_cube)
5           & self.weak_pre(states)
6           ).forsome(gamma_cube)
```

$$\text{Pre}_{\langle \Gamma \rangle}(Z) = \exists a_{\Gamma}. \exists a_{\neg \Gamma} \text{Pre}_{S_{\langle \Gamma \rangle}}(Z)$$

3. Example: CTL model checking

```
1 def main(modelPath):
2     init_nusmv()
3     fsm = BddFsm.from_filename(modelPath)
4     propDb = glob.prop_database()
5     for prop in propDb:
6         if prop.type == propTypes['CTL']:
7             spec = prop.exprcore
8             violating = fsm.init & ~eval_ctl(fsm, spec)
9             print('Specification', str(spec), 'is', str(violating.is_false()))
10            # We could generate counter-examples here
11            deinit_nusmv()
```

```
1 def eval_ctl(fsm, spec):
2 ...
3 elif spec.type == parser.IFF:
4     left = eval_ctl(fsm, spec.car, context)
5     right = eval_ctl(fsm, spec.cdr, context)
6     return (left & right) | (~left & ~right)
7 elif spec.type == parser.AF:
8     left = eval_ctl(fsm, spec.car, context)
9     return ~eg(fsm, ~left)
10 ...
```

$$\phi \iff \psi = (\phi \wedge \psi) \vee (\neg \phi \wedge \neg \psi)$$

$$\text{AF } \phi = \neg \text{EG } \neg \phi$$

```
1 def fixpoint(func, start):
2     old = start
3     new = func(start)
4     while old != new:
5         old = new
6         new = func(old)
7     return old
8 def eg(fsm, phi):
9     return fixpoint(lambda Z: phi & fsm.pre(Z),
10                    BDD.true(fsm.bddEnc.DDmanager))
```

$$\text{EG } \phi = \nu Z. \phi \wedge \text{Pre}(Z)$$

5. Evaluation: CTL model checking

same algorithm (C/Python)

Model	NuSMV (seconds)	PyNuSMV (seconds)
counter	0	0
mutex	0.001	0.009
dme1	0.275	0.288
gas-nq7	8.913	11.027
msi_wtrans	23.285	23.652
dme1-16	61.246	64.733
ftp3	75.607	78.771
key10	100.614	103.606

1-6% overhead on large models

6. Advantages and drawbacks

+ Access to **all NuSMV functionalities** through lower or upper interface.

+ **High-level language** with garbage collection, etc.

+ **Less code to write:** access to model parsing, evaluating simple expressions, etc.

- **Stick to NuSMV architecture**, not written to be used as a library.

- No unified **error management**.

7. Future work

■ Supporting **more NuSMV features:** SAT, LTL model checking, model simulation, etc.

■ A **homogeneous error management** in the upper interface with Python exceptions.