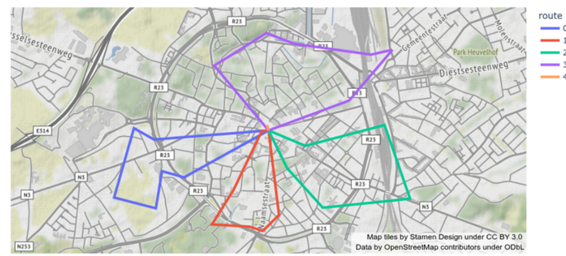
Things we underestimated while developing the CPMpy constraint modelling library

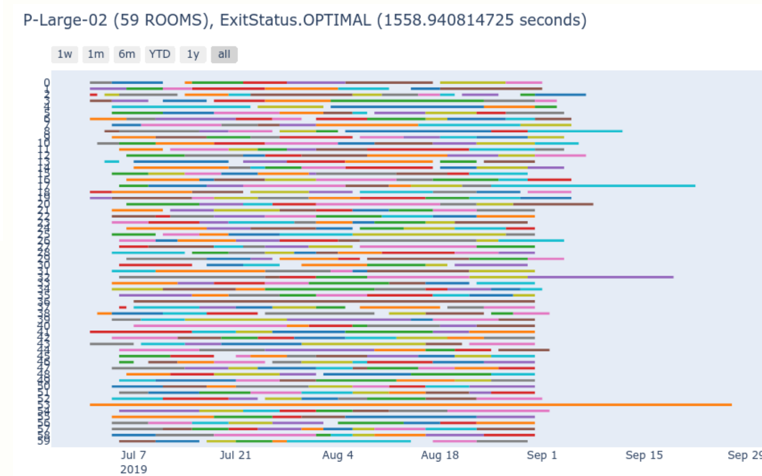
Prof. Tias Guns <tias.guns@kuleuven.be> @TiasGuns
Ignace Bleukx
Wout Vanroose
Emilio Gamba
Jo Devriendt
Dimos Tsouros
Helene Verhaeghe
Ahmed K.A. Abdullah
Maxime Mulamba
Victor Bucarey
Jayanta Mandi
Nicholas Decleyre

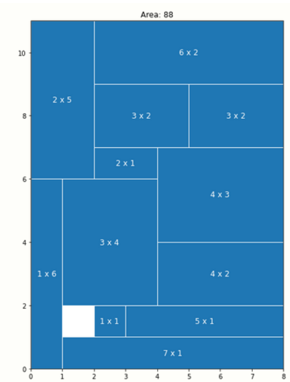
Constraint programming

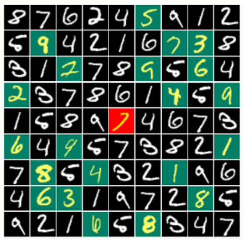
"Solving combinatorial optimisation problems"

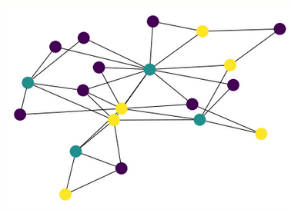
- Vehicle Routing
- Scheduling
- Packing
- Other combinatorial problems









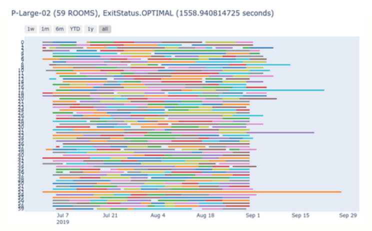


[Solved and visualized with the CPMpy constraint solving library]

Constraint solving paradigm

Model + Solve

Decision variables
Constraints
Objective function



Modeling Tools

	Modeling language	System language	Solver interfacing	Data wrangling
MiniZinc	MiniZinc	C++	Text-based (flatzinc)	minizinc-python
Conjure	Essence	Haskell	Text-based (essence’)	Jupyter notebooks
Savile Row	Essence’	Java	Java	Java?
Picat	Picat	C	C	Picat?
CPMpy	Python	Python	Python	Python

CPMpy vision

A top-down effort to make CP technology more accessible to AI researchers and users in general.

CPMpy vision

A top-down effort to make CP technology more accessible to AI researchers and users in general.

- Easy integration with Python ML & visualisation libraries
  => decision variables are numpy arrays
- Solver-independent, connect to Python ecosystems
  => to CP, SMT, ILP, SAT and BDD python packages
- Incremental solving and direct access to solvers
- Out-of-the-box UNSAT cores, hyperparam tuning, etc

Solvers

CPMpy only interfaces to Python APIs

Key principle: solver can implement any subset of expressions!

Solvers can also choose to:

- Support assumptions or not
- Be incremental or not
- Expose own solver parameters

Currently:
- ortools
- pysat
- minizinc
- gurobi
- pySDD
- Z3
- Exact

Wishlist: GCS, CPOptimiser, Choco, Cplex, Mistral2, Gecode, ...

Things we underestimated...

Supporting typical constraint models

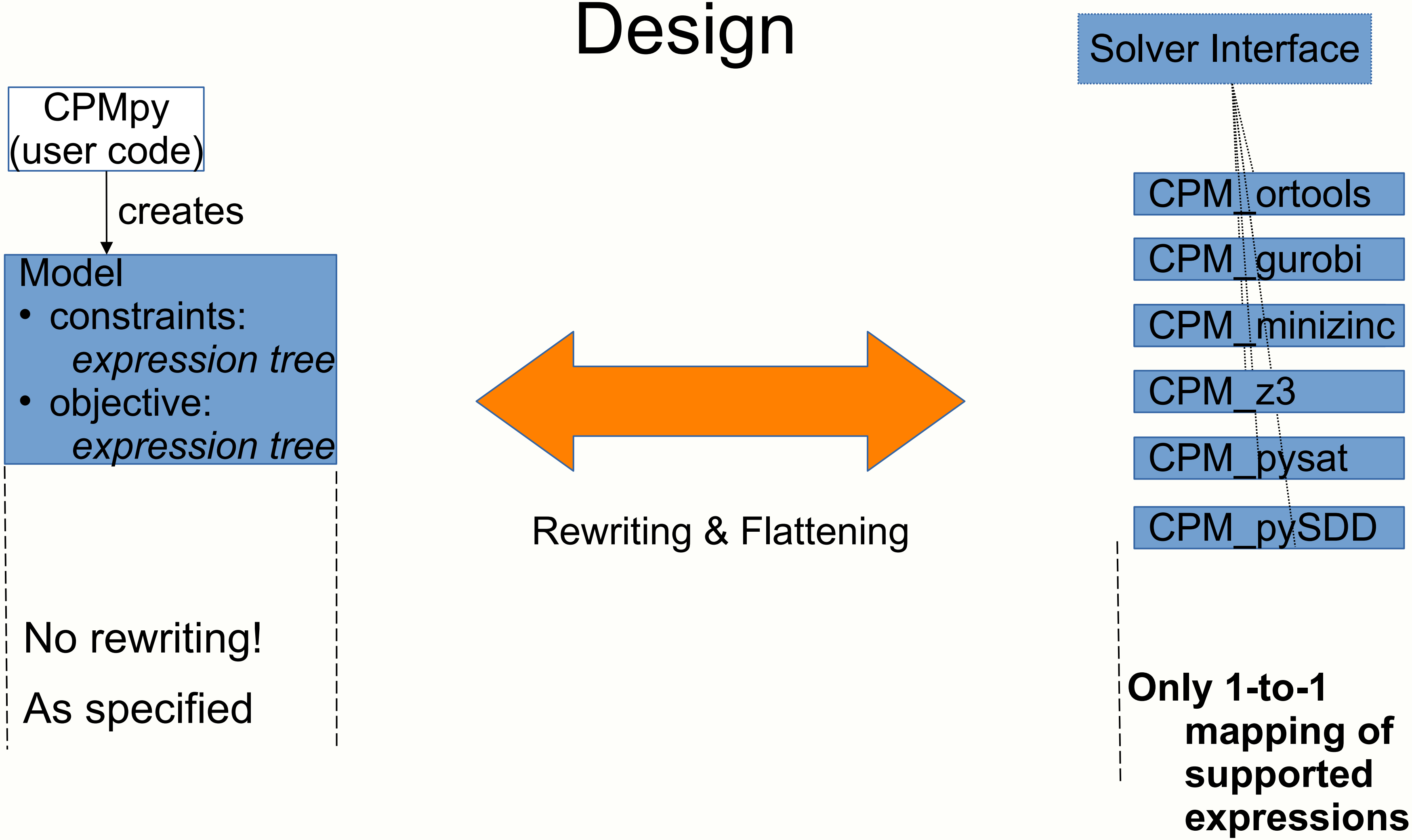
Vs

Supporting typical constraint models

Vs

Supporting all valid input

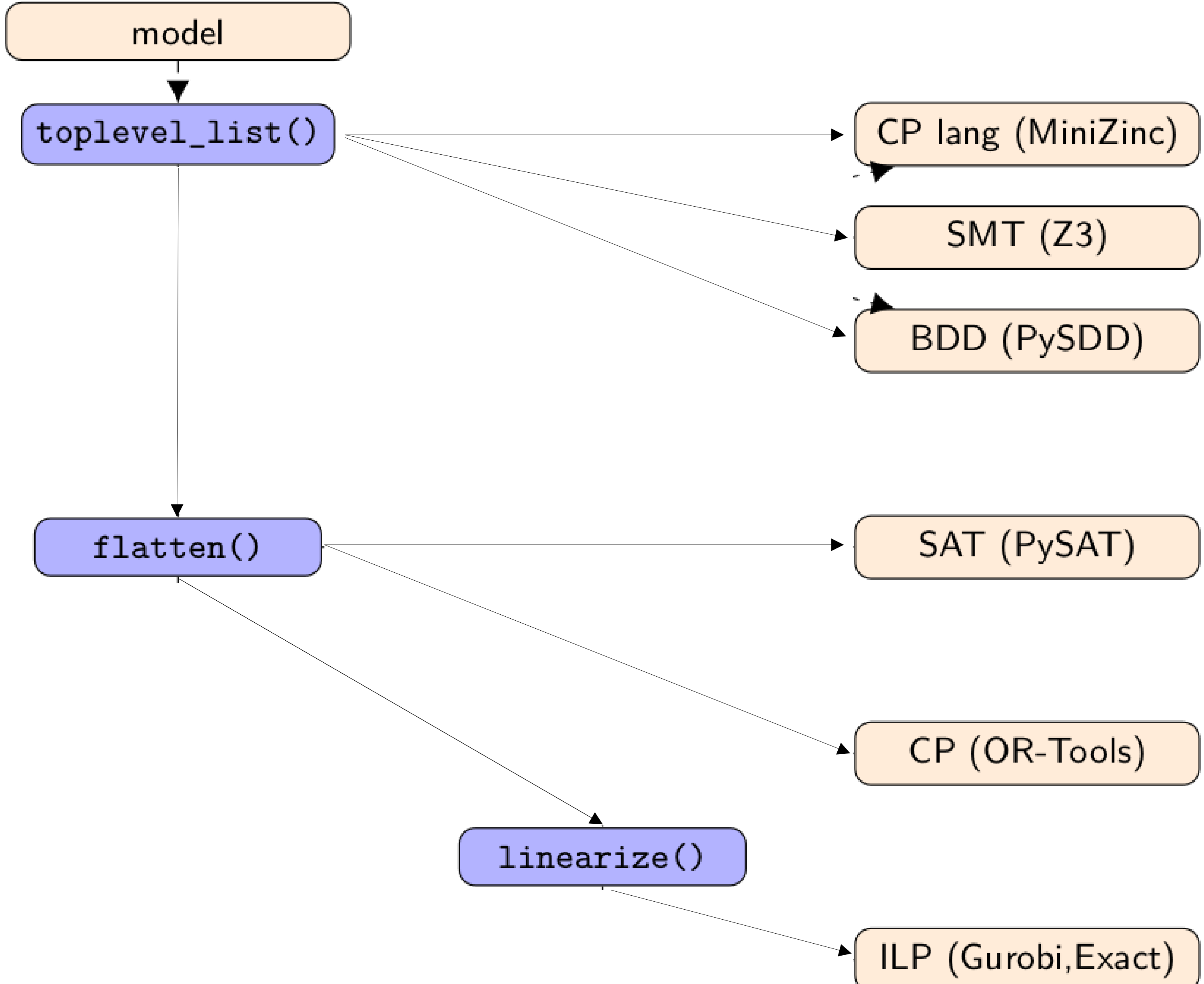
Design


CPMpy
(user code)
creates
Model
• constraints:
expression tree
• objective:
expression tree
No rewriting!
As specified
Rewriting & Flattening
Solver Interface
CPM_ortools
CPM_gurobi
CPM_minizinc
CPM_z3
CPM_pysat
CPM_pySDD
Only 1-to-1
mapping of
supported
expressions

Underestimation 1

- *Flattening* is **central** to CP and SAT

- but SMT and BDDs accept nested input
  - a.implies(b) | (c & ~(a|d))
  - no need to create auxiliary variables!


model
toplevel_list()
CP lang (MiniZinc)
SMT (Z3)
BDD (PySDD)
flatten()
SAT (PySAT)
CP (OR-Tools)
linearize()
ILP (Gurobi,Exact)

- *Flattening* is **central** to CP and SAT

- but SMT & BDDs accept nested input
  - `a.implies(b) | (c & ~(a|d))`
  - no need to create auxiliary variables!

  - they don’t support global constraints though...

Underestimation 2

Global constraints are **central** to CP

- Just decompose them, well studied in CP!

But any expression in CPMpy can be *nested* in another expression

- If your language supports a global constraint,
  it must also support the reified global constraint
- Solvers don't support reified global constraints…

- Reified global == reification of the decomposition?

On the reification of global constraints, 2013.
N. Beldiceanu, M. Carlsson, P. Flener, J. Pearson

"most global constraints can be reformulated as a conjunction of total function constraints together with a constraint that can be easily reified"

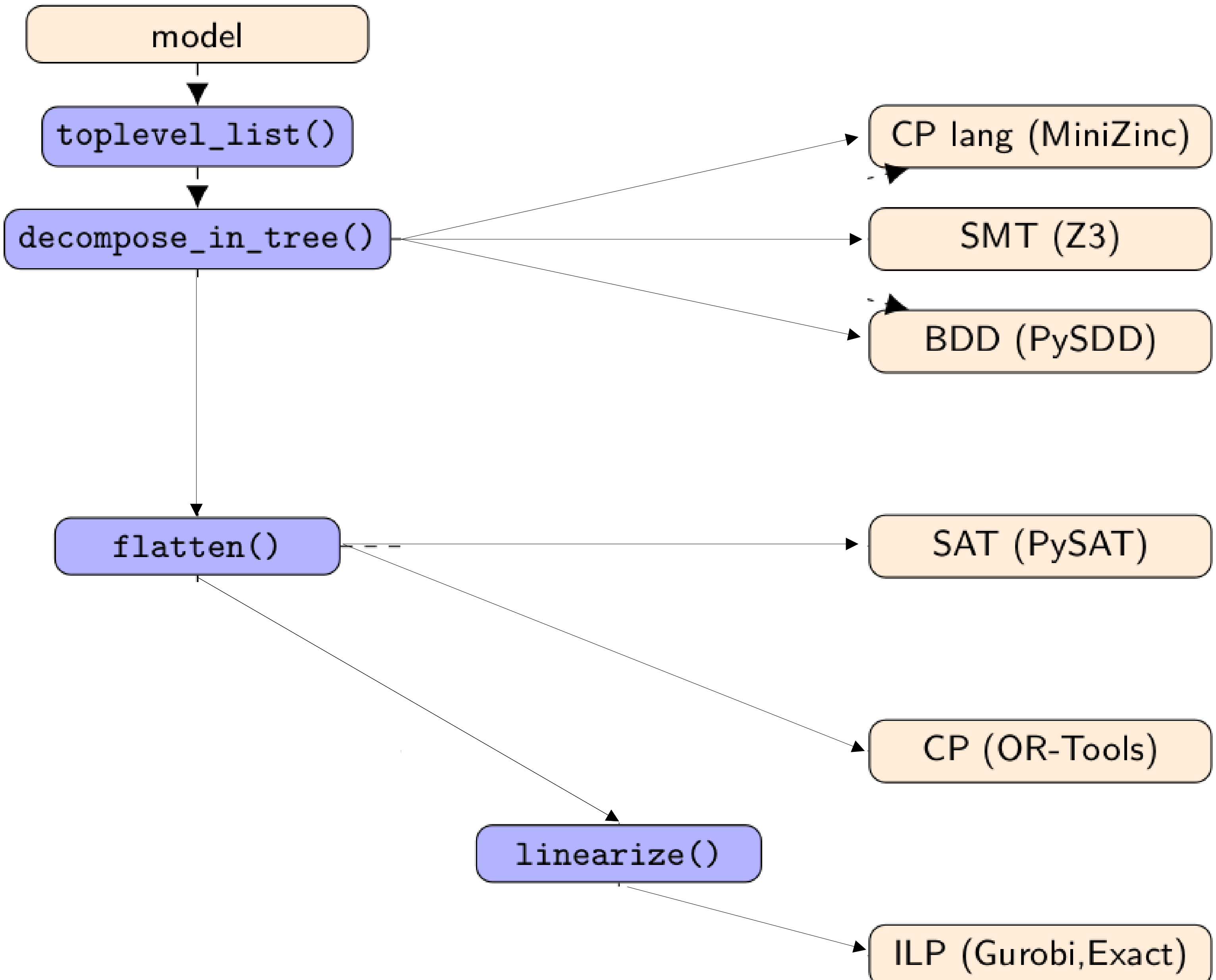
Key issue: decompositions may define auxiliary variables.

Example: Circuit(*nodes*): creates *successor* variables

Our approach:

G.decompose() = (reifiable cons, defining cons)

- Toplevel G: `reifiable & defining`
- Reified, bv ↔ G: `(bv ↔ reifiable) & defining`


model
toplevel_list()
decompose_in_tree()
flatten()
linearize()
CP lang (MiniZinc)
SMT (Z3)
BDD (PySDD)
SAT (PySAT)
CP (OR-Tools)
ILP (Gurobi,Exact)

Underestimation 3

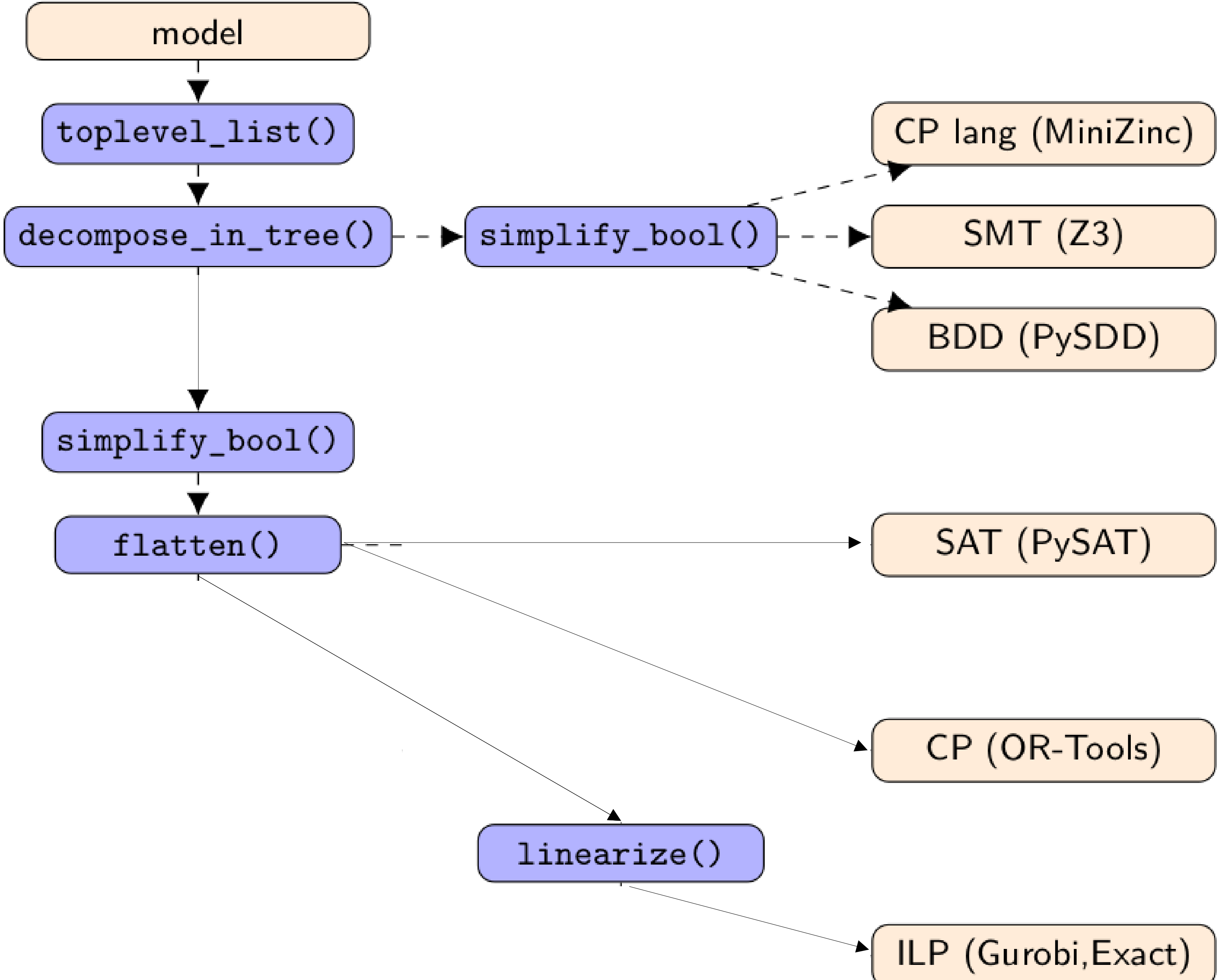
Bool and integer coercions

Flexibility in the language vs strict typing in the solvers.

- No automatic coercion? Or automatically coerce all?
  What about BV == ~IV?

=> Bool can be used as int (common use case, e.g. sum(bvs), not the other way around

Solvers require strict typing. When coercing, stay in Bool space if you can
(e.g. BV1 == 0, BV1 + BV2 >= 1)


model
toplevel_list()
decompose_in_tree()
simplify_bool()
simplify_bool()
flatten()
linearize()
CP lang (MiniZinc)
SMT (Z3)
BDD (PySDD)
SAT (PySAT)
CP (OR-Tools)
ILP (Gurobi,Exact)

Underestimation 4

Negation

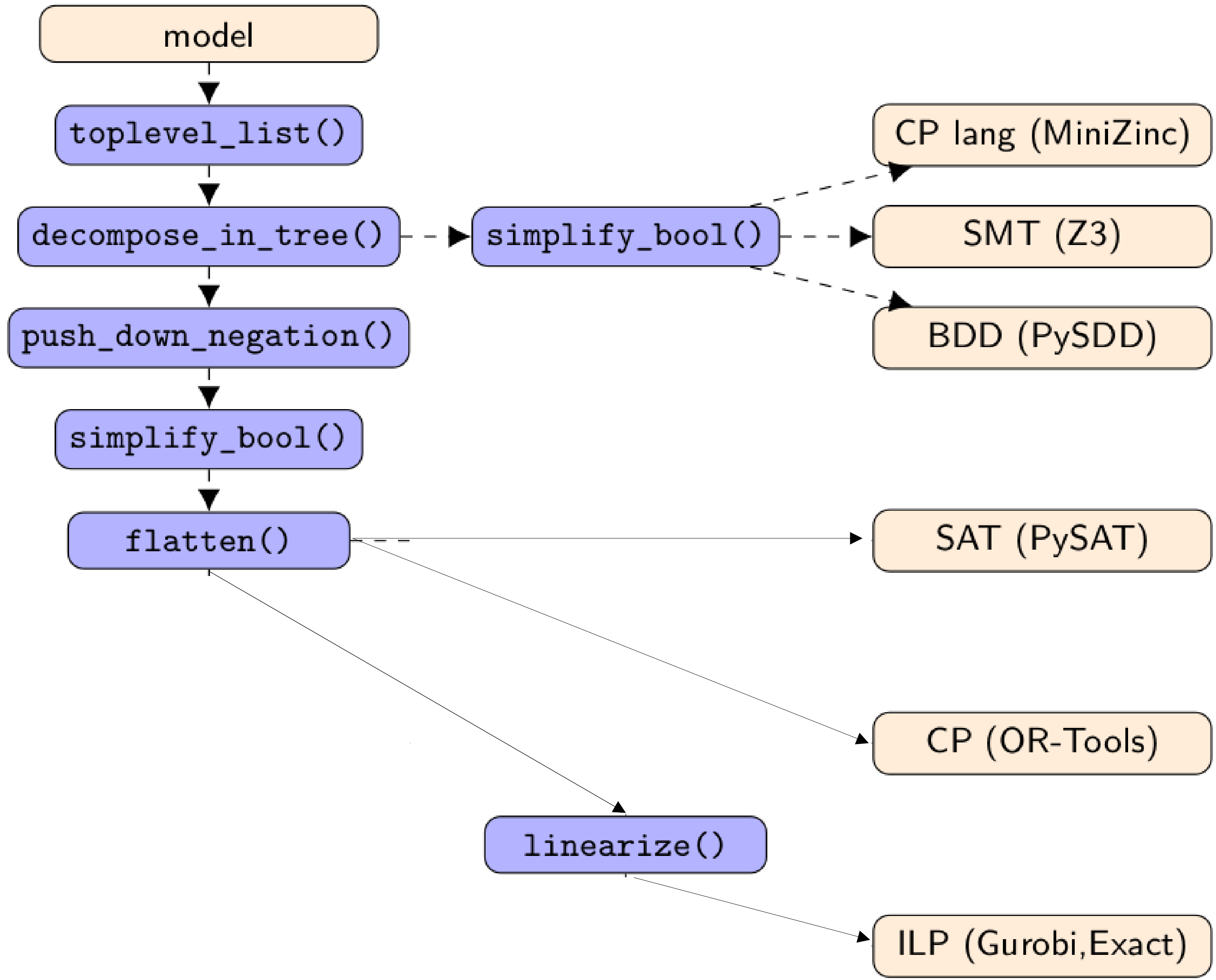
- Just push it down to leafs of expression tree...

→ what about global constraints? OK with reifiable,defining

→ but don't push all negation down for SMT/BDD…

→ avoid introducing unnecessary auxiliary variables in general

So push down early,
do know that later transformations can re-introduce negation…
(creates loop)



model
toplevel_list()
decompose_in_tree()
simplify_bool()
push_down_negation()
simplify_bool()
flatten()
linearize()
CP lang (MiniZinc)
SMT (Z3)
BDD (PySDD)
SAT (PySAT)
CP (OR-Tools)
ILP (Gurobi,Exact)

Underestimation 4

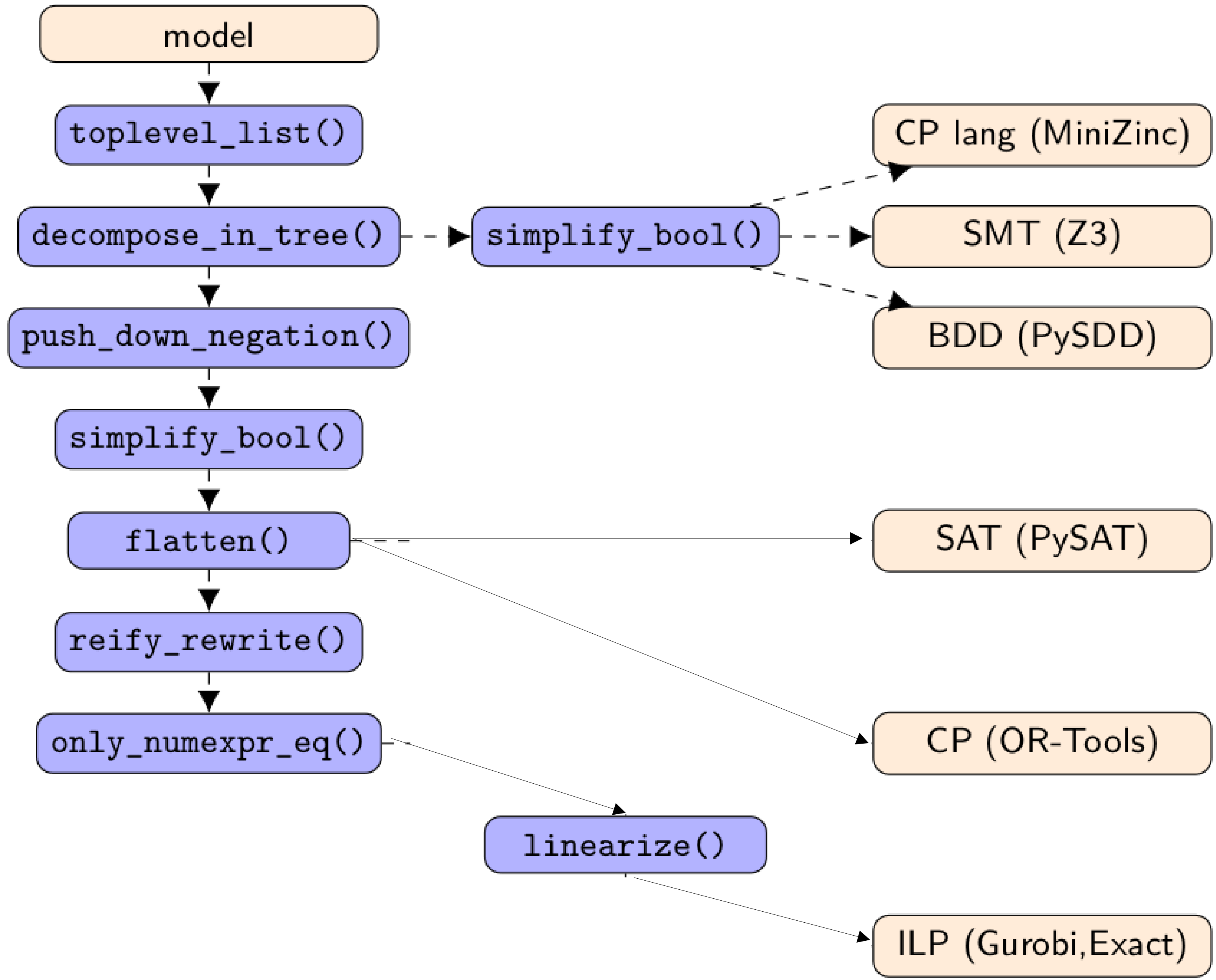
Global constraints again!

- Is Abs(x) == y a global constraint?
- Is Abs(x) >= y a global constraint?
- Is BV ↔ Abs(x) >= a global constraint?

Our solution: Abs(x) is a <u>global function</u>

To solvers that support Abs(x) == y, we rewrite each of the above as:

- Abs(x) == y
- Abs(x) == tmp & tmp >= y
- Abs(x) == tmp & BV ↔ tmp >= a

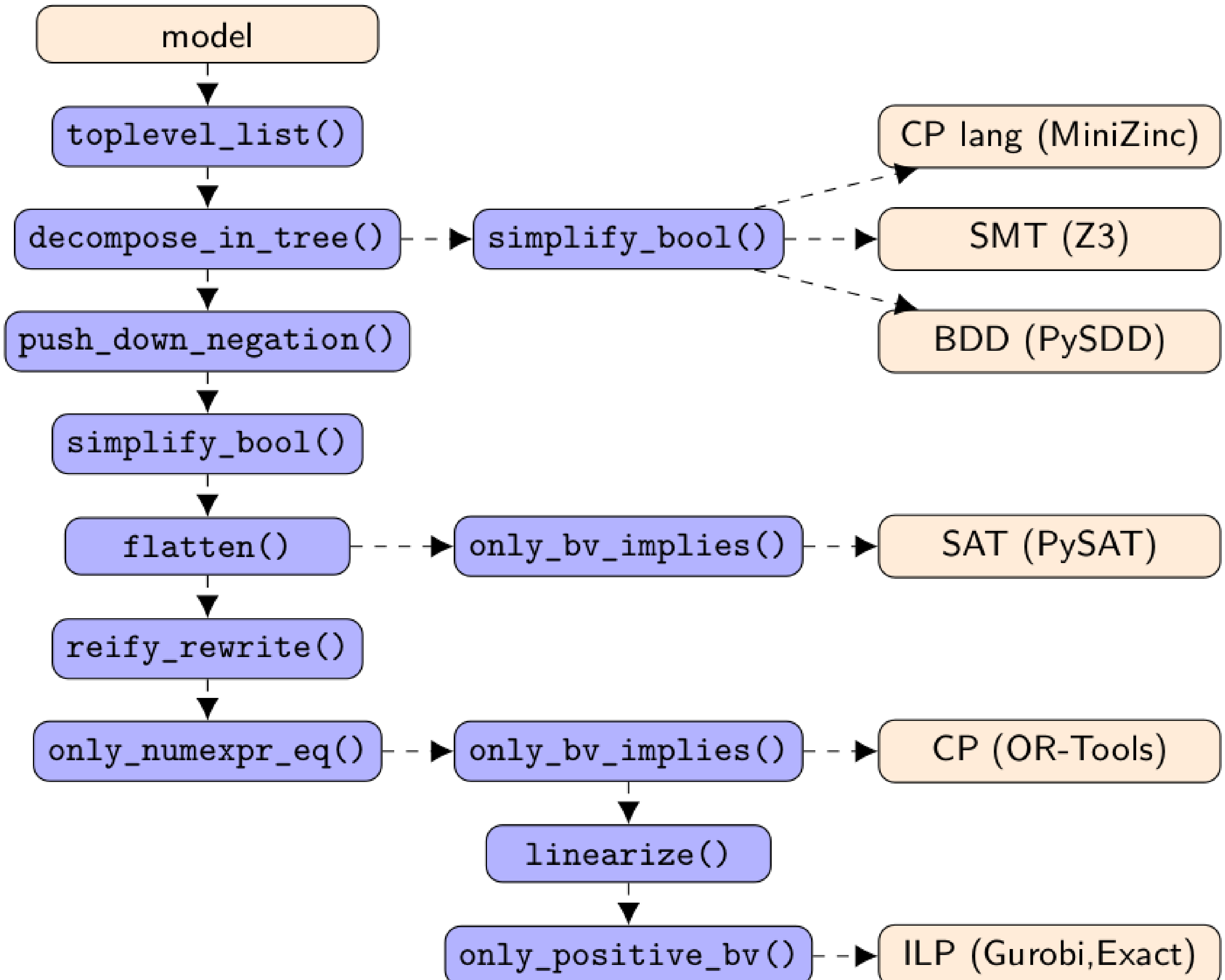

model
toplevel_list()
decompose_in_tree()
simplify_bool()
push_down_negation()
simplify_bool()
flatten()
reify_rewrite()
only_numexpr_eq()
linearize()
CP lang (MiniZinc)
SMT (Z3)
BDD (PySDD)
SAT (PySAT)
CP (OR-Tools)
ILP (Gurobi,Exact)

Underestimation 5

Linearisation

- ILP modeling is so similar, and yet so different...

- Custom decompositions (e.g. of AllDifferent, Xor, Circuit)
- Avoid Big-M formulations where possible
- lhs/rhs of expressions versus canonical linear constraint
- negated Boolean variable versus *negative* Boolean var in sum


model
toplevel_list()
decompose_in_tree()
simplify_bool()
push_down_negation()
simplify_bool()
flatten()
only_bv_implies()
reify_rewrite()
only_numexpr_eq()
only_bv_implies()
linearize()
only_positive_bv()
CP lang (MiniZinc)
SMT (Z3)
BDD (PySDD)
SAT (PySAT)
CP (OR-Tools)
ILP (Gurobi,Exact)

Underestimation 6

<u>Semantics:</u> which solutions are valid, how many in total

What semantics do the solvers follow? E.g.

- For ‘element’ global constraint?
  => assumes total (index variable is bounded to array) or not?

- For integer division
  => exact division, floor division, fractional division?

Partial functions… (for now: we assume all are total)

Underestimation 7

How can you be sure everything is correct?

All cases you can think of?

For all possible expression trees across all solvers (CP,MIP,SMT,SAT)?

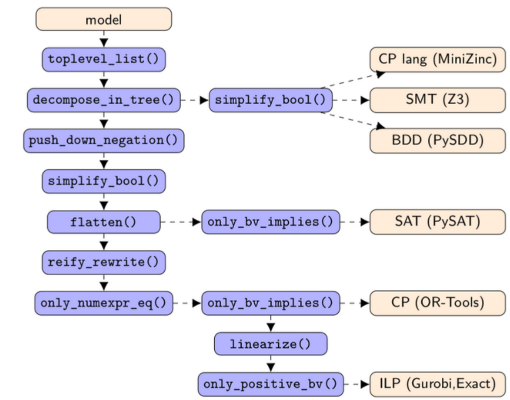
=> Automated fuzztesting!

Supporting typical constraint models

Vs

Supporting all valid input

Conclusion / discussion



- Typical model vs all models
- Keep as much structure as solver supports

- Reify everything?
- FuzzTest everything?
- Efficiency?
- Partial functions?

Extra thanks to Hakan Kjellerstrand for initial testing, Ruben Kindt for initial fuzztesting!